

Decidable Logics for Transductions and Data Words*

Luc Dartois
Université libre de Bruxelles

Emmanuel Filiot
Université libre de Bruxelles
F.R.S.-FNRS

Nathan Lhote
Université libre de Bruxelles
LaBRI, Université de Bordeaux

Abstract—We introduce a logic, called \mathcal{L}_T , to express properties of transductions, i.e. binary relations from input to output (finite) words. In \mathcal{L}_T , the input/output dependencies are modeled via an *origin function* which associates with any position of the output word, the input position from which it originates. The logic \mathcal{L}_T can express all MSO-definable functions, and is incomparable with MSO-transducers for relations. Despite its high expressive power, we show, among other interesting properties, that \mathcal{L}_T has decidable satisfiability and equivalence problems.

The transduction logic \mathcal{L}_T is shown to be expressively equivalent to a logic for data words, \mathcal{L}_D , up to some bijection from transductions with origin to data words (the origin of an output position becomes the data of that position). The logic \mathcal{L}_D , which is interesting in itself and extends in expressive power known logics for data words, is shown to have decidable satisfiability.

I. INTRODUCTION

Language theory and applications. The theory of regular languages of finite words is rich and robust, founded on three important pillars: computation (automata theory), algebra (theory of monoids) and logic (monadic second-order logic). It has been successfully extended to other classes of languages [1], [2] and structures, such as trees [3] and infinite words [4].

A well-known application of the logic-automata connection for (finite and infinite) word languages is the theory of model-checking, in the domain of computer-aided verification [5], where system computations are modeled by automata, and system specifications are written in high-level formalism, a logic. Specifications are then turned into automata and model-checking then reduces to language inclusion. Many logics and automata models have been proposed in this domain, with different modeling features, expressiveness, and complexities with respect to the model-checking problem, contributing to the success of automata and logic techniques in verification.

Theory of transductions. In this paper, we go beyond languages and consider binary relations of finite words, aka transductions. They relate input words over some finite alphabet Σ to output words over some alphabet Γ . E.g., the transduction τ_{double} duplicates each input symbol ($(ab, aabb) \in \tau_{\text{double}}$), and τ_{shuffle} associates a word with all its permutations ($(ab, ab), (ab, ba) \in \tau_{\text{shuffle}}$). A transduction is said to be functional if it is a (partial) function. The theory of transductions is not as advanced as the theory of languages, but important

results exist. *At the computational level*, (1-way) finite automata with *outputs*, called transducers, have been studied for long [6], [7]. They have been extended to 2-way transducers by allowing the input head to move in both directions. Among the most important results are the decidability of equivalence [8] and the closure under composition [9] of deterministic 2-way transducers. More recently, a 1-way deterministic model of transducers with word registers, called streaming string transducers, has been introduced and shown to be equivalent to deterministic 2-way transducers [10]. *At the algebraic level*, a canonical object, called bimachines, has been introduced for functions definable by 1-way transducers [11], which has been recently the basis of a more general study of definability problems for such functions [12]. Finally, *at the logical level*, only one logical formalism is known, called MSO-transducers (MSOT), which was interestingly shown to be equivalent, for the class of functions, to deterministic 2-way transducers [13].

MSOT have been introduced by Courcelle [14] in a more general context, as a formalism to define functions of arbitrary relational structures. The main idea is to define the n -ary predicates of the output structure, by formulas of monadic second-order logic (MSO) with n free first-order variables, interpreted over the input structure. Moreover, the input structure can be copied a fixed number of times and the MSO formulas are parameterised by the copies their free variables belong to. By taking a linear-order \leq and unary predicates for labels, one obtains the signature of words and hence MSOT can define functional word transductions. MSOT have been extended with non-determinism (called NMSOT) to define relations in general. Non-determinism is obtained by having a fixed tuple of free monadic second-order variables that can be used in any of the formulas of the NMSOT. NMSOT are known to correspond to non-deterministic streaming string transducers [15] but not to non-deterministic 2-way transducers [13].

Need for a new logic. Despite the appealing connection between MSOT, deterministic 2-way transducers and deterministic streaming string transducers (which all define the class of so-called regular functions), it turns out that MSOT and NMSOT are not satisfactory as a specification language for transductions, in the sense that they are still too operational, thus precluding novel applications such as model-checking data-processing systems. For instance, specifying the order of the output word as MSO-formulas over the input word

* This work is supported by the ARC project Transform (Wallonia-Brussels Federation), the FNRS PDR project Flare (J013116F), and the French ANR project ExtStream (ANR-13-JS02-0010).

somehow forces the designer to describe the left-right moves of a 2-way machine running on the input word. Even in the non-deterministic setting, once some interpretation of the extra parameters has been fixed, then the resulting transduction is functional, and one is forced to fully specify it. In a model-checking scenario, if one focuses on some critical property rather than on the full system specification, it is desirable to have a formalism that leaves parts of the system unspecified. Thus, having a high level of non-determinism is crucial in this context. It is however impossible in NMSOT, a trivial example being that nothing is specified: the universal transduction of non empty words $\Sigma^+ \times \Gamma^+$ is not definable in NMSOT.

In this paper, our objective is to propose a logic tailored to transductions, with the following requirements: it should (i) be expressive, by that we mean it should at least capture the robust class of regular functions, (ii) offer a high level of non-determinism (to leave parts of the system unspecified), (iii) be decidable with respect to satisfiability.

The proposed logic \mathcal{L}_T for transductions. The logic we define in this paper is based on a simple idea: a pair of words of a transduction can be seen as a single structure, with two orders \leq_{in} and \leq_{out} on respectively the input and output positions, and unary predicates for the labels. However, such a structure is not rich enough to express interesting dependencies between input and output positions, and therefore we also assume the existence of an *origin* mapping, which maps any output position to some input position, intuitively the position from which it originates. As noticed in [16], any known transducer model, including NMSOT, implicitly produce some origin information when translating an input word. For instance, a transition (p, a, q) with output bc of a finite-state transducer not only translate a into bc , but also produces the origin of b and c as the current position. In this paper, a transduction is not only a set of pairs of words, but it is a set of *productions* $(u, (v, o))$ where u is the input word, v is the output word, and o maps any position of v to a position of u . E.g. τ_{shuffle} can be naturally extended with origin:

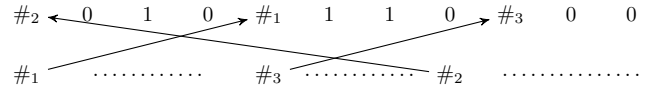


Fig. 1. Two productions of τ_{shuffle}

A production $(u, (v, o))$ can be seen as a structure with the orders \leq_{in} , \leq_{out} , unary predicates for the labels, and some origin function symbol o . It turns out that even first-order logic is undecidable over such structures. We propose a restriction, the logic \mathcal{L}_T , defined intuitively as follows. We take the two-variable fragment of first-order logic over this signature (i.e. only two variable names can be used in the formulas), but extend it with all MSO-definable binary predicates over \leq_{in} and Σ , hence being able only to talk about the input word.

Example. Suppose some server receives task requests from a fixed number k of clients. For simplicity we assume that there is only one task, which consists in realizing some function $t : \{0, 1\}^* \rightarrow \{0, 1\}^*$. To request for task processing, Client $i \in \{1, \dots, k\}$ sends the word $\#_i m$, where $\#_i$ is

a client identifier, and $m \in \{0, 1\}^*$ is the message to be processed by task t . The server gets as input a list of task requests $\#_{i_1} m_1 \dots \#_{i_n} m_n$, and treats them in any order, thus outputting a word $\#_{i_{\sigma(1)}} t(m_{\sigma(1)}) \dots \#_{i_{\sigma(n)}} t(m_{\sigma(n)})$ for some permutation σ of $\{1, \dots, n\}$. Now, one would like to verify the following property: every request is processed exactly once. Therefore, we are not interested here on whether the task t is realised correctly, but rather on whether every request is treated. This could be expressed by requiring the system to output $\#_i$ whenever it processes a request $\#_i m$ of Client i , and then requiring that the origin mapping defined by this input/output behaviour is bijective and label-preserving. A possible example of such a correct behaviour is:



where the dots represent any word in $\{0, 1\}^*$, whose origins are not specified (and so not depicted). The property can be expressed in \mathcal{L}_T by saying that o is a label-preserving bijection between hash positions (using special quantifiers over input and outputs positions):

$$\forall^{\text{in}} x \bigwedge_{i=1}^k \#_i(x) \rightarrow (\exists^{\text{out}} y \ o(y) = x \wedge \#_i(y)) \wedge \forall^{\text{out}} x \forall^{\text{out}} y \bigwedge_{i,j=1}^k (\#_i(x) \wedge \#_j(y) \wedge o(x) = o(y)) \rightarrow x = y$$

If one wants to verify that the system processes the requests in the same order as the one it gets them, then we express that the input/output orders are preserved between hash positions:

$$\forall^{\text{out}} x \forall^{\text{out}} y \bigwedge_{i,j=1}^k (\#_i(x) \wedge \#_j(y) \wedge x \leq_{\text{out}} y) \rightarrow o(x) \leq_{\text{in}} o(y)$$

None of the properties described before are definable in NMSOT, because NMSOT forces to fully specify the result of a transduction (hence here we must also specify what the results $t(m_i)$ should be). Also, τ_{shuffle} is not NMSOT-definable.

Contributions on \mathcal{L}_T . We show, and it is our main result, that \mathcal{L}_T has decidable satisfiability problem. As a result, the equivalence problem, is decidable for transductions (with origin information) defined in \mathcal{L}_T . On the expressiveness side, we prove that any \mathcal{L}_T -transduction has regular domain, that \mathcal{L}_T can express any function definable in MSOT, and is incomparable with NMSOT. We also show that the functionality problem (whether any input word is mapped to at most one pair (v, o)) and the bounded-origin problem (whether any input position is the origin of at most k output positions, for some constant k) are decidable. The decidability of \mathcal{L}_T is obtained via a reduction to a new logic for data words.

Connection with data words and the logic \mathcal{L}_D . A transduction is non-erasing if for any of its production $(u, (v, o))$, every u position is the origin of at least some v position. We show that every \mathcal{L}_T -transduction can be turned into a non-erasing one while preserving some interesting properties, including satisfiability. There is a simple bijection between non-erasing transductions and so called *typed data words*. Let $u = a_1 \dots a_n \in \Sigma^*$ and $v = b_1 \dots b_m \in \Gamma^*$. Any

production $(u, (v, o))$ can be encoded as the sequence $w = (b_1, a_{o(1)}, o(1)) \dots (b_m, a_{o(m)}, o(m))$. E.g., the left production of Fig. 1 is encoded as $(a, a, 1)(c, c, 3)(a, a, 4)(b, b, 2)$. The sequence w is called a *typed data word*, the integers being the data, and the symbol $a_{o(i)}$ being the type of the data $o(i)$. In a typed data word $w = (\gamma_1, \sigma_1, d_1) \dots (\gamma_k, \sigma_k, d_k)$, if two triples have the same datum $d_i = d_j$, then they are required to have the same type $\sigma_i = \sigma_j$. The data define a total preorder \preceq on the positions of w by $i \preceq j$ if $d_i \leq d_j$. Then, typed data words can be seen as structures with the linear-order \leq on positions, the preorder \preceq , and unary predicates for labels and types. The logic \mathcal{L}_T translates into a logic we call \mathcal{L}_D on typed data word structures, defined as follows. It is the two-variable first-order logic over Γ and \leq , extended with all binary MSO-predicates over \preceq and Σ with the following restriction: in MSO-predicates, monadic second-order variables X range over sets of positions closed for data equality (if $x \in X$ and y has the same data as x , then $y \in X$). Hence, MSO predicates can really be seen as formulas which quantifies over data and sets of data, and can express properties such as “there is an even number of data of some given type t ”.

The logic \mathcal{L}_D strictly extends the logic $\text{FO}^2[\Gamma, \preceq, S_d, \leq]$ over (untyped) data words (where S_d is the successor over data), for which the satisfiability problem is known to be EXPSPACE-C [17]. Still, we show that \mathcal{L}_D has decidable satisfiability problem. The proof extends the techniques of [17] which further reduce the problem to a constraint satisfaction problem on the two-dimensional plane. This extension is non-trivial as the MSO predicates induce new constraints that are handled using query automata running vertically on the plane. Unlike in [17], we also use an automata approach to recognise sequences of types $\sigma_1 \dots \sigma_n$ which can be extended to typed data words $(\gamma_1, \sigma_1, d_1) \dots (\gamma_n, \sigma_n, d_n)$ that are models of the formula, thus proving regularity of such sequences. The automata approach allows us to get more results, among which the decidability of the data boundedness problem (is there some bound k such that any data occurs at most k times in any model of the formula). We believe that the results on \mathcal{L}_D are of independent interest for the data word community.

Other related works. The study of transductions with origin was first initiated in [16] where the goal was to obtain an algebraic characterisation of (functional) transductions with origin definable in MSOT. Transductions with origin have been extended to finite trees in [18] in which the complexity of the equivalence problem is analysed for various transducer classes.

Logics for data words have been studied extensively over the past few years [17], [19], [20], but have been focused so far on establishing a decidability frontier for the two-variable fragment of first-order logic with combinations of various predicates (on data: equality, linear-order, successor, and on positions: linear-order, successor). See [17] for a summary. To the best of our knowledge, it is the first time full MSO predicates have been considered to speak about the data. Through the encoding of transductions with origin as data words, some undecidability and decidability results

for transductions are consequences of known results for data words. It is the case for instance of the undecidability of the logic $\text{FO}^2[\Sigma, \Gamma, \leq_{\text{in}}, \leq_{\text{out}}, S_{\text{out}}]$ where S_{out} is the successor over output positions by using [20], and the decidability of $\text{FO}^2[\Sigma, \Gamma, \leq_{\text{in}}, \leq_{\text{out}}]$ by using [17].

Organisation of the paper. In Section II, we introduce the logic \mathcal{L}_T and give the main results. In Section III, we give the connection between transductions and data words, and introduce the logic \mathcal{L}_D . Section IV is devoted to proving the decidability of \mathcal{L}_D . Finally, in Section V, we prove the decidability of \mathcal{L}_T and gives a few consequences of the proof techniques adopted for the decidability of \mathcal{L}_D . Most proofs are only sketched but can be found in the appendix.

II. LOGIC WITH ORIGIN FOR TRANSDUCTIONS

A. Words and Transductions

Given a (finite) alphabet Σ , a word u of length $n \geq 0$ (denoted by $|u|$) is a function from $\{1, \dots, n\}$ into Σ . When $n = 0$, the domain of this function is empty and we denote by ϵ this function, called the empty word. We define $\text{dom}(u) = \{1, \dots, n\}$ and for all $i \in \text{dom}(u)$, i is called the i th position of u , and $u(i)$ the i th symbol of u (or letter). The set of (non-empty) words over Σ is denoted by Σ^* (Σ^+).

Let Σ and Γ be two disjoint alphabets. An *origin-free transduction* is a subset of $\Sigma^* \times \Gamma^*$. A *production with origin* (or production for short) from Σ to Γ is a pair $(u, (v, o))$ such that $u \in \Sigma^+$, $v \in \Gamma^*$ and o is a (total) mapping from $\text{dom}(v)$ to $\text{dom}(u)$, called the *origin mapping*. We denote by $\mathcal{PR}(\Sigma, \Gamma)$ the set of productions from Σ to Γ . A *transduction with origin* (or just transduction) τ from Σ to Γ is a set of productions $(u, (v, o)) \in \mathcal{PR}(\Sigma, \Gamma)$. Intuitively, a pair $(u, (v, o))$ means that the word u is translated into v , and that any position $i \in \text{dom}(v)$ has been “produced” while processing position $o(i)$ of u . Somehow, a transduction with origin not only says what is translated into what, but also how. As noticed in [16], most transducer models in the literature can be interpreted as machines for transductions *with* origin. We say that τ is functional (or is a function) if for all u , there is at most one pair (v, o) such that $(u, (v, o)) \in \tau$, and rather denote it by f instead of τ . The *domain* of a transduction τ is the set of words u such that there exists (v, o) such that $(u, (v, o)) \in \tau$. Finally, the *origin-free projection* of τ is the origin-free transduction $\{(u, v) \mid \exists (u, (v, o)) \in \tau\}$.

Example 1. Let Σ be a finite alphabet, and let $\Gamma = \{\bar{\sigma} \mid \sigma \in \Sigma\}$ (remind that we require the alphabets to be disjoint). By extension, we also write $\bar{u} \in \Gamma^*$ for the annotated version of any word $u \in \Sigma^*$. We give several examples of transductions (with origin) from Σ to Γ . First, consider the identity f_{id} defined for all $u \in \Sigma^*$ by $f_{\text{id}}(u) = (\bar{u}, \text{id})$ where id is the identity function over $\text{dom}(u)$. As an example, consider the following production $(u, (v, o))$ in f_{id} :

input u	a	b	c	a	a	b	
	\uparrow	\uparrow	\uparrow	\uparrow	\uparrow	\uparrow	origin o
output v	\bar{a}	\bar{b}	\bar{c}	\bar{a}	\bar{a}	\bar{b}	

The next transduction τ_{shuffle} associates an input word to all its permutations, i.e. $(u, (v, o)) \in \tau_{\text{shuffle}}$ iff o is a label-preserving bijection from $\text{dom}(v)$ to $\text{dom}(u)$. For example, consider the following two productions (with same input word):



B. FO and MSO Logics

A *signature* \mathbb{S} is a (possibly infinite) set of predicate symbols P with an arity $\text{ar}(P) \in \mathbb{N}$ and a (possibly infinite) set of functional symbols f with an arity $\text{ar}(f) \in \mathbb{N}^+$ (functions of arity 0 are called constants). A (finite) *structure* \mathcal{M} over a signature \mathbb{S} consists of a finite domain D together with an interpretation $\cdot^{\mathcal{M}}$ of each predicate symbol P by an $\text{ar}(P)$ -ary relation¹ $P^{\mathcal{M}} \subseteq D^{\text{ar}(P)}$, and of each functional symbol f by an $\text{ar}(f)$ -ary (total) function² from $D^{\text{ar}(f)}$ into D .

Let $\mathcal{X}_1, \mathcal{X}_2$ be two disjoint countable sets of first- and (monadic) second-order variables respectively. The monadic-second order logic over \mathbb{S} (denoted by $\text{MSO}[\mathbb{S}]$) is the set of formulas generated by the following grammar:

$$\phi ::= \exists x \phi \mid \exists X \phi \mid \phi \wedge \phi \mid \neg \phi \mid X(t) \mid P(t_1, \dots, t_n) \mid \top$$

where $x \in \mathcal{X}_1$, $X \in \mathcal{X}_2$, P is a predicate of \mathbb{S} of arity n , and t, t_1, \dots, t_n are terms over the function symbols. The atom $X(t)$ means that t belongs to X (X is viewed as a unary predicate). Universal quantifiers $\forall x \phi$ and $\forall X \phi$ are naturally defined by $\neg \exists x \neg \phi$ and $\neg \exists X \neg \phi$. We also define the false formula $\perp \equiv \neg \top$. We may write $\phi(x_1, \dots, x_n, X_1, \dots, X_m)$ to emphasise that the first- and second-order free variables of a formula ϕ are respectively x_1, \dots, x_n and X_1, \dots, X_m .

Without defining the semantics of MSO (the reader can see, e.g., [1]), let us mention that, over a structure \mathcal{M} with domain D , first-order variables are interpreted by elements of D while second-order variables are interpreted by subsets of D .

The *first-order logic* fragment over \mathbb{S} (denoted by $\text{FO}[\mathbb{S}]$), is the set of $\text{MSO}[\mathbb{S}]$ formulas without quantification over second-order variables nor free second-order variables. The *two-variable* fragment of first-order logic, denoted by $\text{FO}^2[\mathbb{S}]$, is the restriction of $\text{FO}[\mathbb{S}]$ to formulas in which only two variables appear (but can be reused). The *existential MSO* fragment ($\exists \text{MSO}[\mathbb{S}]$) is the set of $\text{MSO}[\mathbb{S}]$ formulas of type $\exists X_1 \dots \exists X_n \phi$ where ϕ is in $\text{FO}[\mathbb{S}, X_1, \dots, X_n]$, where each X_i is viewed as a unary predicate. Similarly, the two-variable fragment of $\exists \text{MSO}$, denoted by $\exists \text{MSO}^2[\mathbb{S}]$ is the set of formulas of type $\exists X_1 \dots \exists X_n \phi$ where ϕ is in $\text{FO}^2[\mathbb{S}, X_1, \dots, X_n]$.

C. FO and MSO Logics for Transductions

Any production $(u, (v, o)) \in \mathcal{PR}(\Sigma, \Gamma)$ is seen as a structure \mathcal{M} over the signature $\mathcal{T}_{\Sigma, \Gamma}$ composed of unary predicates

¹With the convention that $D^0 = \{\emptyset\}$ (empty tuple), a predicate symbol of arity 0 is either interpreted by false (empty set) or true ($\{\emptyset\}$)

²functions of arity 0 are constant functions, and therefore are interpreted by a single element of D .

$\delta(x)$, for all $\delta \in \Sigma \cup \Gamma$, the two order predicates \leq_{in} and \leq_{out} , and the function symbol \mathbf{o} , interpreted as follows:

- the domain of \mathcal{M} is $\text{dom}(u) \uplus \text{dom}(v)$
- $\sigma^{\mathcal{M}} = \{i \in \text{dom}(u) \mid u(i) = \sigma\}$ for any $\sigma \in \Sigma$,
- $\gamma^{\mathcal{M}} = \{i \in \text{dom}(v) \mid v(i) = \gamma\}$ for any $\gamma \in \Gamma$,
- $\leq_{\text{in}}^{\mathcal{M}}$ is the natural linear order on $\text{dom}(u)$,
- $\leq_{\text{out}}^{\mathcal{M}}$ is the natural linear order on $\text{dom}(v)$,
- $\mathbf{o}^{\mathcal{M}}$ is the function³ $i \mapsto o(i)$ if $i \in \text{dom}(v)$, and $i \mapsto i$ if $i \in \text{dom}(u)$.

We also use the predicates $=$, $<_{\text{in}}$ and $<_{\text{out}}$, which are all definable in the logics that we consider. For an $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ -formula ϕ , we write $(u, (v, o)) \models \phi$ instead of $\mathcal{M} \models \phi$, and implicitly assume that $(u, (v, o))$ is given as a structure over $\mathcal{T}_{\Sigma, \Gamma}$. We may write $\text{MSO}[\Sigma, \Gamma, \leq_{\text{in}}, \leq_{\text{out}}, \mathbf{o}]$ for $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ to underline which symbols are used in the signature.

Any $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ sentence ϕ defines a transduction $\llbracket \phi \rrbracket$, which is the set of productions $(u, (v, o))$ such that $(u, (v, o)) \models \phi$. We say that a transduction τ is definable in $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ if $\tau = \llbracket \phi \rrbracket$ for some sentence $\phi \in \text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$. We also say that an origin-free transduction τ' is definable in $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ if there exists an (origin) transduction τ definable in $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ and whose origin-free projection is τ' .

Example 2. We first define several macros that will be useful throughout the paper. The formula $\text{in}(x) \equiv x \leq_{\text{in}} x$ (resp. $\text{out}(x) \equiv x \leq_{\text{out}} x$) hold true if x belongs to the input word (resp. output word). Now for $\alpha \in \{\text{in}, \text{out}\}$, we define the guarded quantifiers $\exists^{\alpha} x \phi$ and $\forall^{\alpha} x \phi$ as shortcuts for $\exists x \alpha(x) \wedge \phi$ and $\forall x \alpha(x) \rightarrow \phi$ (note that $\neg \exists^{\alpha} x \phi$ is equivalent to $\forall^{\alpha} x \neg \phi$). We could also define similarly guarded second-order quantifiers. Even if it is not necessary, we will often use guarded quantifiers for the sake of formula readability. The following formulas express that the origin mapping is respectively an injection, surjection and bijection from output positions to input positions:

$$\begin{aligned} \phi_{\text{inj}} &\equiv \forall^{\text{out}} x, y (\mathbf{o}(x) = \mathbf{o}(y)) \rightarrow x = y \\ \phi_{\text{surj}} &\equiv \forall^{\text{in}} x \exists^{\text{out}} y x = \mathbf{o}(y) \\ \phi_{\text{bij}} &\equiv \phi_{\text{inj}} \wedge \phi_{\text{surj}} \end{aligned}$$

We can use it to define the transduction τ_{shuffle} of Example 1:

$$\phi_{\text{shuffle}} \equiv \phi_{\text{bij}} \wedge \forall^{\text{out}} x \bigwedge_{\sigma \in \Sigma} \sigma(\mathbf{o}(x)) \rightarrow \bar{\sigma}(x)$$

If we also require the origin mapping to be order-preserving, we get a formula defining the identity transduction f_{id} :

$$\begin{aligned} \phi_{\text{order-pres.}} &\equiv \forall^{\text{out}} x \forall^{\text{out}} y x \leq_{\text{out}} y \rightarrow \mathbf{o}(x) \leq_{\text{in}} \mathbf{o}(y) \\ \phi_{\text{id}} &\equiv \phi_{\text{shuffle}} \wedge \phi_{\text{order-pres.}} \end{aligned}$$

Given a class \mathcal{C} of $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ formulas, we say that \mathcal{C} is *decidable* if the following satisfiability problem is decidable: given an $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ formula ϕ , does there exist a production $(u, (v, o))$ from Σ to Γ such that $(u, (v, o)) \models \phi$. In other words, it asks whether $\text{dom}(\llbracket \phi \rrbracket) \neq \emptyset$. It turns out that the logic $\text{MSO}[\mathcal{T}_{\Sigma, \Gamma}]$ is undecidable, even if restricted to FO:

³To make its domain total, \mathbf{o} is interpreted also on $\text{dom}(u)$, by the identity.

Proposition 3. *Over transductions, the logic $FO[\Sigma, \Gamma, \leq_{in}, \leq_{out}, \mathcal{O}]$ is undecidable.*

Proof. It is a consequence of a stronger result: Prop. 5. \square

However, if one restricts to the two-variable fragment, one regains decidability. This can be shown by using the decidability from [17] of the logic FO^2 with one linear-order and one total pre-order interpreted over data words, and a correspondence between data words and transductions given in Section III, or as a particular case of a more expressive logic that we define in Section II-D.

Proposition 4. *Over transductions, the logic $FO^2[\Sigma, \Gamma, \leq_{in}, \leq_{out}, \mathcal{O}]$ is decidable.*

It is known that the successor predicate cannot be expressed in the two-variable first-order logic. However, if one adds this predicate to the signature, the first-order logic is undecidable, even if restricted to the two-variable fragment. The proof of this result is an adaptation to transductions of the undecidability, over data words, of FO^2 with a linear-order and successor predicates over positions, and a linear-order on data [20].

Proposition 5. *Over transductions, the logic $FO^2[\Sigma, \Gamma, \leq_{in}, \leq_{out}, S_{out}, \mathcal{O}]$ is undecidable.*

D. The decidable logic \mathcal{L}_T for transductions with origin

The logic \mathcal{L}_T extends $FO^2[\Sigma, \Gamma, \leq_{in}, \leq_{out}, \mathcal{O}]$, which is decidable, with any binary predicate definable in $MSO[\leq_{in}, \Sigma]$ which is only allowed to quantify over input positions, in order to capture regular properties of the input words and, as we will see, of the output words as well, while preserving decidability. This extension will also be expressive enough to capture functional transductions definable in MSOT [14], [13].

We let $MSO_{bin}[\leq_{in}, \Sigma]$ be the set of n -ary predicates, for $n \leq 2$, written $\{\phi\}$ where ϕ is an $MSO[\leq_{in}, \Sigma]$ -formula with n free variables and the quantifications are over the input. We use the brackets $\{$ and $\}$ to explicit the fact that $\{\phi\}$ is a predicate symbol and not just an MSO formula. Over a production $(u, (v, o))$, an n -ary predicate $\{\phi\}$ is interpreted as an n -ary relation over u , naturally defined by the interpretation of the MSO formula on u .

The logic, denoted by \mathcal{L}_T , is the two-variable fragment of first-order logic over the output symbol predicates, the linear-order \leq_{out} , and all predicates in $MSO_{bin}[\leq_{in}, \Sigma]$, i.e.

$$\mathcal{L}_T := FO^2[\Gamma, \leq_{out}, \mathcal{O}, MSO_{bin}[\leq_{in}, \Sigma]]$$

It should be clear that \mathcal{L}_T is a fragment of $MSO[\Sigma, \Gamma, \leq_{out}, \leq_{in}, \mathcal{O}]$ and as such, there is no need to define its semantics.

Example 6. Order-preservation Preservation of the input/output orders through origin is expressed by the \mathcal{L}_T -formula $\forall^{out} x, y (x \leq_{out} y) \rightarrow \{x' \leq_{in} y'\}(\mathcal{O}(x), \mathcal{O}(y))$.

Note that we could equivalently replace x' and y' by any variable (even x and y), without changing the semantics: the formula $x' \leq_{in} y'$ defines a binary relation on the input word, which is used as an interpretation of the predicate $\{x' \leq_{in} y'\}$. To ease the notations, any predicate $\{\phi\}(t_1, t_2)$ where ϕ

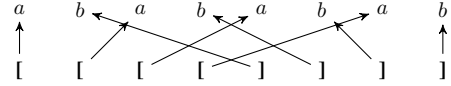
has two free variables x_1 and x_2 may be sometimes written $\{\phi[x_1/t_1, x_2/t_2]\}$, i.e. ϕ in which the x_i have been substituted by t_i . We keep the brackets $\{$ and $\}$ to emphasise the fact it is a binary MSO formula which speaks about the input word. Hence, the latter formula can be written as:

$$\phi_{order-pres.} \equiv \forall^{out} x, y (x \leq_{out} y) \rightarrow \{\mathcal{O}(x) \leq_{in} \mathcal{O}(y)\}$$

Note that all formulas of Example 2 can be translated into \mathcal{L}_T , modulo brackets (see Appendix).

Identity on a regular language Since the MSO predicates only speak about the input word, it is tempting to think that the expressive power on the output is restricted to FO^2 . It turns out that thanks to the origin mapping, the regular properties of the input words transfer to the output word. As an example, any regular domain restriction of f_{id} is \mathcal{L}_T -definable. Let $L \subseteq \Sigma^*$ be a regular language and let $\phi_L \in MSO[\leq_{in}, \Sigma]$ be a sentence defining L . Then, the restriction $f_{id}|_L$ is defined by $\phi_{id} \wedge \{\phi_L\}$.

Dyck languages Consider the (origin-free) transduction $(ab)^n \mapsto [^n]^n$ where the output alphabet consists of $\Gamma = \{[,]\}$. By taking any bijective origin mapping such that each $[$ is mapped to a and each $]$ to b , e.g. as follows:



then one obtains a transduction definable by some \mathcal{L}_T -formula ϕ . Indeed, the language $(ab)^*$ being regular, it is definable by an $MSO[\leq_{in}, \Sigma]$ -formula $\phi_{(ab)^*}$. Then, ϕ is defined by:

$$\begin{aligned} \phi_{dom} &\equiv \{\phi_{(ab)^*}\} \\ \phi_{range} &\equiv \forall^{out} x \forall^{out} y [(x) \wedge](y) \rightarrow x \leq_{out} y \\ \phi_{lab} &\equiv \forall^{out} x [(x) \rightarrow \{a(\mathcal{O}(x))\} \wedge](x) \rightarrow \{b(\mathcal{O}(x))\}] \\ \phi &\equiv \phi_{dom} \wedge \phi_{range} \wedge \phi_{bij} \wedge \phi_{lab} \end{aligned}$$

More generally, one can associate with any word $(ab)^n$ the set of all well-parenthesised words of length n over Γ . Thus, the range of this transduction is a Dyck language over $\{[,]\}$. This transduction is realised by the formula $\phi_{dom} \wedge \phi_{bij} \wedge \phi_{lab} \wedge \phi'$ where ϕ' expresses that for any two output positions x and y , if $o(y)$ is the successor of $o(x)$ and is labelled b , then $x \leq_{out} y$:

$$\phi' \equiv \forall^{out} x \forall^{out} y \{S_{in}(\mathcal{O}(x), \mathcal{O}(y)) \wedge b(\mathcal{O}(y))\} \rightarrow x \leq_{out} y$$

This is correct due to the fact that a word w over Γ is well-parenthesised if, and only if, for all its prefixes, the number of $[$ is greater than the number of $]$, and equal on w . This is ensured by the formula ϕ' which forces the production $[$ of an a to appear before the production $]$ of its successor b .

Remark 7. According to the previous examples, one can express in \mathcal{L}_T the transduction τ_1 defined as the shuffle over the language a^*b^* , and also $\tau_2 : (ab)^n \mapsto a^n b^n$. Hence the composition $\tau_2 \circ \tau_1 : a^n b^n \mapsto a^n b^n$ has a non-regular domain. However, as we will see in Section V, the domain of an \mathcal{L}_T -transduction is always regular, which means that \mathcal{L}_T -transductions are not closed under composition.

One of our main result is the decidability of \mathcal{L}_T :

Theorem 8. *Over transductions, the logic \mathcal{L}_T is decidable.*

The proof of this theorem relies on a reduction to a data word logic, called \mathcal{L}_D and introduced in Section III, which has the same expressiveness as \mathcal{L}_T , modulo encodings of transductions as data words and conversely, and which is shown to be decidable in Section IV. From the technics developed in the decidability proof, we extract several interesting results such as regularity of domains of \mathcal{L}_T -definable transductions, decidability of functionality and of origin-boundedness. These results are given in Section V.

An important and desirable consequence of the decidability of \mathcal{L}_T is the decidability of the equivalence problem, which asks, given two \mathcal{L}_T -sentences ϕ_1, ϕ_2 , whether $\llbracket \phi_1 \rrbracket = \llbracket \phi_2 \rrbracket$. This amounts to check whether $\neg(\phi_1 \leftrightarrow \phi_2)$ is unsatisfiable.

Corollary 9. *The equivalence problem for \mathcal{L}_T -definable transductions is decidable.*

By extending the alphabets, the decidability of \mathcal{L}_T can be trivially extended to that of $\exists\text{MSO}^2$:

Corollary 10. *Over transductions, the logic $\exists\text{MSO}^2[\Gamma, \leq_{\text{out}}, \mathbf{o}, \text{MSO}_{\text{bin}}[\leq_{\text{in}}, \Sigma]]$ is decidable.*

Note that $\exists\text{MSO}^2[\Gamma, \leq_{\text{out}}, \mathbf{o}, \text{MSO}_{\text{bin}}[\leq_{\text{in}}, \Sigma]]$ is not closed under negation, and therefore we do not get decidability of equivalence for this logic as a consequence of this corollary.

Finally, we conclude this section by showing that \mathcal{L}_T lies somehow at the decidability frontier, if one wants the power of MSO on the input (which is necessary to capture Courcelle's MSO transductions). Adding just the successor relation over the output words leads indeed to undecidability.

Proposition 11. *Over transductions, $\text{FO}^2[\Gamma, \leq_{\text{out}}, S_{\text{out}}, \mathbf{o}, \text{MSO}_{\text{bin}}[\leq_{\text{in}}, \Sigma]]$ is undecidable.*

Proof. As shown by Prop. 4, the logic $\text{FO}^2[\Sigma, \Gamma, \leq_{\text{in}}, \leq_{\text{out}}, S_{\text{out}}, \mathbf{o}]$ is already undecidable. \square

E. Expressiveness of \mathcal{L}_T

In this section, we compare our logic to MSO-transducers, as introduced by Courcelle in [14], which define (partial) functions from logical structures over some signature to logical structures over a possibly different signature. Casted to the word signature with unary predicates $\sigma(x)$ for labels and with a linear order \leq between positions, it defines a logical formalism for word-to-word transductions, which is known to be equivalent to deterministic 2-way finite state transducers [13]. While these formalisms have been defined without origin semantics, they intrinsically bear origin information [16]. We first show that any transduction definable by some MSO-transducer is definable in \mathcal{L}_T .

In an MSO-transducer, the output word structure is defined by taking a fixed number k of copies of the input word domain. Nodes of these copies can be filtered out by MSO formulas with one free first-order variable. In particular, the nodes of the c -th copy are the input positions that satisfy some given MSO formula $\phi_{\text{pos}}^c(x)$. The output label predicates $\gamma(x)$ and

the order predicate $x \leq y$ of the output structure are defined by MSO formulas with respectively one and two free first-order variables, interpreted over the input structure. Formally, an MSO-transducer (MSOT) is a tuple $T =$

$$\left(k, \phi_{\text{dom}}, (\phi_{\text{pos}}^c(x))_{c=1}^{\leq k}, (\phi_{\gamma}^c(x))_{c=1, \gamma \in \Gamma}^{\leq k}, (\phi_{\leq}^{c,d}(x, y))_{c,d=1}^{\leq k} \right)$$

where $k \in \mathbb{N}$ and the formulas $\phi_{\text{dom}}, \phi_{\text{pos}}^c, \phi_{\gamma}^c$ and $\phi_{\leq}^{c,d}$ are MSO $[\Sigma, \leq]$ -formulas. We refer the reader for instance to [21], [13], [22] for the (origin-free) semantics of these transducers. The origin semantics is obtained by adding the origin mapping which associates with any copies c of input position x , the input position x . We call MSO-transduction a transduction definable in this formalism.

Example 12. As an example, we show how to define the transduction $f : u \mapsto uu$ with origin mapping $o(i) = ((i - 1) \bmod |u|) + 1$ by an MSO-transducer. We use two copies ($k = 2$), the position formulas are all true and the label formulas are $\phi_{\gamma}^c(x) = \gamma(x)$, $c = 1, 2$. The order formulas are $\phi_{\leq}^{i,i}(x, y) = x \leq y$ for $i = 1, 2$, $\phi_{\leq}^{1,2}(x, y)$ is true and $\phi_{\leq}^{2,1}(x, y)$ is false.

This transduction is definable in \mathcal{L}_T as follows. We first define copy formulas $c_1(x) = \forall y \ y <_{\text{out}} x \rightarrow \{\mathbf{o}(x) \neq \mathbf{o}(y)\}$ and $c_2(x) = \neg c_1(x) \wedge \forall y \ y <_{\text{out}} x \rightarrow (c_1(y) \vee \{\mathbf{o}(x) \neq \mathbf{o}(y)\})$. Then the order is enforced by the order formula: $\psi_{\leq}(x, y) = x \leq_{\text{out}} y \leftrightarrow ((c_1(x) \wedge c_2(y)) \vee (\bigvee_{i=1,2} c_i(x) \wedge c_i(y) \wedge \{\mathbf{o}(x) \leq_{\text{in}} \mathbf{o}(y)\}))$. Now the function f is defined in \mathcal{L}_T by

$$\forall^{\text{out}} x (c_1(x) \vee c_2(x)) \wedge \bigvee_{\gamma \in \Gamma} \gamma(x) \leftrightarrow \{\gamma(\mathbf{o}(x))\} \wedge \forall^{\text{out}} y \psi_{\leq}(x, y)$$

The latter encoding can be (effectively) generalised to any MSO-transduction:

Theorem 13. *Any MSO-transduction is \mathcal{L}_T -definable.*

We conjecture the converse of Theorem 13 is true, i.e. whether for all (origin-free) functions $f : \Sigma^* \rightarrow \Gamma^*$ definable in \mathcal{L}_T , f is an MSO-transduction. As a good sign towards this conjecture, we observe that functions definable in \mathcal{L}_T are linear-size increase, just as MSO-definable functions. We indeed show, in Section V, that for all \mathcal{L}_T -formula ϕ (not necessarily defining a function), there exists a linear-size increase uniformisation of it.

An appealing consequence of Theorem 13 and the decidability of \mathcal{L}_T is the decidability of the *model-checking* problem for expressive classes of transductions. Given a system realizing some functional transduction f (with origin), defined in some expressive operational model such as a deterministic 2-way transducer T_f , and given some specification ϕ_S defined in \mathcal{L}_T , one can decide whether $T_f \models \phi_S$, i.e. for all inputs $u \in \text{dom}(f)$, $(u, f(u)) \in \llbracket \phi_S \rrbracket$. It suffices to convert T_f into some MSO-transducer (based on the equivalence between deterministic 2-way transducers and MSO-transducers [13]) which in turn is converted into some \mathcal{L}_T -formula ϕ_f , thanks to Thm. 13, and then to test the (un)satisfiability of $\phi_f \wedge \neg \phi_S$.

MSO-transductions have been extended to relations (called NMSO-transductions), by using a set of monadic second-

order parameters X_1, \dots, X_n . Every formulas of NMSO-transductions can use X_1, \dots, X_n as free variables. Once an interpretation of these variables as sets of positions is fixed, the transduction becomes functional. Therefore, the maximal number of output words for the same input word is bounded by the number of interpretations of X_1, \dots, X_n .

Proposition 14. *The class of NMSO- and \mathcal{L}_T -transductions are incomparable. Any NMSO-transduction is expressible in $\exists \text{MSO}^2[\Gamma, \leq_{\text{out}}, \mathbf{0}, \text{MSO}_{\text{bin}}[\leq_{\text{in}}, \Sigma]]$.*

Indeed consider the transduction τ_{even} defined as the set of pairs (u, vv) where v is a subword of u of even length. It is definable in NMSO by using a parameter X which defines the input positions that are kept. Then, τ_{even} is defined almost as in Example 12: The position formulas $\phi_{\text{pos}}^c(x)$ are just changed to $X(x)$ and the domain formula states that the predicate X is of even size. τ_{even} is not \mathcal{L}_T -definable as it requires a quantification over the output within the scope of a second order quantification over the input. Conversely, neither the universal transduction $\Sigma^+ \times \Gamma^+$ nor the shuffle are NMSO-definable, due to the bound on the number of images of an input word, while they are \mathcal{L}_T -definable. The inclusion is a trivial extension of Thm. 13 by existentially quantifying the parameters X_i . Going back to model checking, one can then check if all models of an NSST A satisfy an \mathcal{L}_T -formula φ . Indeed since NSST are equivalent to NMSOT, one can construct a formula $\varphi_A \in \exists \text{MSO}^2[\Gamma, \leq_{\text{out}}, \mathbf{0}, \text{MSO}_{\text{bin}}[\leq_{\text{in}}, \Sigma]]$ and check if $\varphi_A \wedge \neg \varphi$ is satisfiable.

III. TYPED DATA WORDS

In this section, we make a connection between transductions and data words, introduce a new logic \mathcal{L}_D for data words, which is shown to be decidable in Section IV. We show that modulo some encodings, the logics \mathcal{L}_D and \mathcal{L}_T are equivalent.

A data word is a word where each position is equipped with a label from a finite alphabet and a value from an infinite domain, its datum. They are especially used to model computational traces in distributed environments, where values correspond to process numbers. Data trees are also used in database theory to model XML documents, where the values model XML attributes. Logics for data words have already been considered (see [19]). In particular, it was proved in [20] that FO with equality over data is undecidable, but FO^2 is. In [17], the authors proved, over an ordered data domain, the decidability of FO^2 equipped with both the order and successor over data, but adding these data predicates is done at the cost of removing the successor for the linear order over positions.

A. Typed data words

We consider here *typed data words*, where our data domain is ordered, and each data also carries a label (type) from a finite alphabet. Formally, a typed data word w over two disjoint alphabets⁴ Γ and Σ is a sequence $w = (\gamma_1, \sigma_{d_1}, d_1) \dots (\gamma_n, \sigma_{d_n}, d_n)$ from $(\Gamma \times \Sigma \times \mathbb{N})^*$ such that

⁴Assuming disjointness is done wlog. We do it for simplifying the proofs.

$\{d_1, \dots, d_n\} = \{1, \dots, m\}$ for some m (i.e. it is closed⁵ by \leq). Note that by definition of typed data words, if $d_i = d_j$, then positions i and j carry the same data type $\sigma_{d_i} = \sigma_{d_j}$ from Σ . We let $\text{dom}(w) = \{1, \dots, n\}$ be the set of positions of w .

The data of a typed data word w induce a total preorder⁶ \preceq on the positions of w defined by $i \preceq j$ if $d_i \leq d_j$. This preorder induces itself an equivalence relation \sim defined by $i \sim j$ iff $i \preceq j$ and $j \preceq i$, which means that the positions i and j carry the same datum.

Given two disjoint alphabets Σ and Γ , we denote by $\mathcal{TDW}(\Sigma, \Gamma)$ the set of typed data words over alphabets Γ and Σ . One might think that the data labels are redundant with the alphabet. However, our goal is to define, similarly to the previous section, a logic where different restrictions apply to the position alphabet and the data types. Thus, having types for data will allow for more expressive power of the logic.

A typed data word $w = (\gamma_1, \sigma_1, d_1) \dots (\gamma_n, \sigma_n, d_n)$ will equivalently be seen as a structure \mathcal{M} over the signature

$$\mathcal{D}_{\Sigma, \Gamma} = \{(\delta(x))_{\delta \in \Sigma \cup \Gamma}, \leq, \preceq\}$$

with the following interpretations:

- $\leq^{\mathcal{M}} \subseteq \text{dom}(w)^2$ is the natural linear order on $\text{dom}(w)$,
- $\preceq^{\mathcal{M}} \subseteq \text{dom}(w)^2$ is the total preorder $i \preceq^{\mathcal{M}} j$ iff $d_i \leq d_j$,
- $\gamma^{\mathcal{M}} = \{i \in \text{dom}(w) \mid \gamma_i = \gamma\}$, for $\gamma \in \Gamma$,
- $\sigma^{\mathcal{M}} = \{i \in \text{dom}(w) \mid \sigma_i = \sigma\}$, for $\sigma \in \Sigma$.

We will also consider the predicate \sim interpreted by the equivalence relation induced by $\preceq^{\mathcal{M}}$ (the predicate \sim will be definable in the data logic we will consider). As for productions, we will write $w \models \phi$, for some logical formula ϕ over $\mathcal{D}_{\Sigma, \Gamma}$ instead of $\mathcal{M} \models \phi$, and implicitly assume that w is given as a structure.

B. The logic \mathcal{L}_D for typed data words

The logic $\text{MSO}[\mathcal{D}_{\Sigma, \Gamma}]$ is known to be undecidable [20] (even the first-order fragment). We define here a fragment of $\text{MSO}[\mathcal{D}_{\Sigma, \Gamma}]$, called \mathcal{L}_D , which will be shown to be decidable.

Intuitively, a formula of \mathcal{L}_D can be seen as an FO^2 formula using the linear order of the positions and some additional binary data predicates. The logic \mathcal{L}_D is indeed built on top of MSO n -ary predicates, for $n \leq 2$, which are allowed to speak only about the data. Precisely, we define $\text{MSO}_{\text{bin}}[\Sigma, \preceq]$ to be the set of n -ary predicates written $\{\phi\}$, for $n \leq 2$, where ϕ is an MSO -formula with n -free first-order variables, over the unary predicates $\sigma(x)$ and the preorder \preceq , with the following semantical restriction⁷: second-order variables are interpreted by \sim -closed sets of positions. Over typed data words (seen as structures over $\mathcal{D}_{\Sigma, \Gamma}$), predicates $\{\phi\}$ are interpreted by n -ary relations on positions, $n \leq 2$, defined by formulas ϕ .

⁵We make this assumption wlog, because the logic we define will only be able to compare the order of data, and therefore cannot distinguish typed data words up to renaming of data, as long as the order is preserved. E.g. $(a, b, 1)(c, d, 3)(e, f, 2)$ and $(a, b, 2)(c, d, 5)(e, f, 4)$ will be undistinguishable by the logic.

⁶We recall that a preorder is a reflexive and transitive relation.

⁷Note that the semantical restriction could also be enforced in the logic by guarding quantifiers $\exists X \psi$ by $\exists X [\forall x \forall y x \in X \wedge y \sim x \rightarrow y \in X] \rightarrow \psi$.

Due to the semantical restriction, formulas in $\text{MSO}_{\text{bin}}[\Sigma, \preceq]$ cannot distinguish positions with the same data and therefore, they can be thought of as formulas which quantify over data and sets of data. As an example, the formula $\forall y \ x \preceq y$ expresses that the data of position x is the smallest data, and it holds true for any x' with the same data. Then, the logic \mathcal{L}_D is defined as the two-variable first-order fragment of $\text{MSO}[\mathcal{D}_{\Sigma, \Gamma}]$ with the MSO-predicates we just defined.

$$\mathcal{L}_D := \text{FO}^2[\Gamma, \leq, \text{MSO}_{\text{bin}}[\Sigma, \preceq]]$$

Example 15. We let $x \sim y$ stands for $x \preceq y \wedge y \preceq x$. First, let us mention that $\text{MSO}_{\text{bin}}[\Sigma, \preceq]$ predicates can express any regular properties about the data, in the following sense. Given a typed data word w , the total preorder \preceq between positions in $\text{dom}(w)$ can be extended to a total order \leq_{\sim} on the equivalence classes of $\text{dom}(w)/\sim$, by $[i]_{\sim} \leq_{\sim} [j]_{\sim}$ if $i \preceq j$. Then, any typed data word induces a word $\sigma_1 \dots \sigma_n \in \Sigma^*$ such that σ_i is the type of the elements of the i th equivalence class, for \leq_{\sim} . Any regular property of these induced words over Σ transfer into a regular property about the data of typed data words (it suffices to replace in the MSO-formula on Σ -words expressing the property, the linear order by \preceq and the equality by \sim). Examples of properties are: n is even, which transfers into “there is an even number of data”, or $\sigma_1 \dots \sigma_n$ contains an even number of $\sigma \in \Sigma$, for some σ , which transfer into “there is an even number of data of type σ ”.

One can also define partial specifications like “occurrences of a symbols are ordered by their data”, which is stated by $\phi_a^{\leq} = \forall x \forall y (a(x) \wedge a(y)) \rightarrow (\{x \preceq y\} \leftrightarrow x \leq y)$.

Our next example is that of a scheduler with a set of ordered processus (whose id are the data) that can be urgent (data type u) or non urgent (data type n). The property that urgent processus are treated first and in id-order is expressed by $\psi = \forall x \forall y \{u(x) \wedge (n(y) \vee x \preceq y)\} \rightarrow x \leq y$. To enforce that processus are only treated once, it suffices to state that no two positions carry the same data: $\psi_{\text{bij}} \equiv \forall x \forall y \{x \sim y\} \rightarrow x = y$.

More generally, all \mathcal{L}_T formulas of Example 6 can be turned into formulas of \mathcal{L}_D , as we will see, allowing us to define, for example, a language of typed data words with $\Gamma = \{I, J\}$ such that its projection on Γ is the set of well-parenthesised words over Γ , or languages of typed data words whose projections on Γ are shuffle closures of regular languages. (the shuffle closure of a regular language L is the set of words $\gamma_{\pi(1)} \dots \gamma_{\pi(n)}$ for π a bijection from $\{1, \dots, n\}$ to $\{1, \dots, n\}$, and $\gamma_1 \dots \gamma_n \in L$. For example, the closure of $(abc)^*$ is the set of words with an equal number of a , b , and c , which is not even context-free.

Our second main result is the next theorem, whose proof is the main goal of Section IV.

Theorem 16. *On typed data words, the logic \mathcal{L}_D is decidable.*

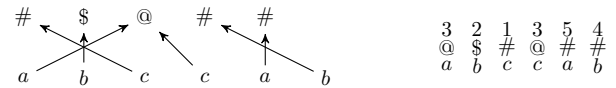
C. From transductions to data words and back

A production $(u, (v, o)) \in \mathcal{PR}(\Sigma, \Gamma)$ is said to be *non-erasing* if o is a surjective function, and an \mathcal{L}_T -formula ϕ is *non-*

erasing if all productions of $\llbracket \phi \rrbracket$ are non-erasing. Satisfiability of \mathcal{L}_T is reducible to satisfiability of non-erasing formula:

Proposition 17. *For any \mathcal{L}_T -formula ϕ there exists a non-erasing \mathcal{L}_T -formula ϕ' such that $\text{dom}(\llbracket \phi \rrbracket) = \text{dom}(\llbracket \phi' \rrbracket)$. In particular, ϕ is satisfiable if, and only if, ϕ' is.*

The reason we consider non-erasing transductions is because there are straightforward encodings of non-erasing productions to typed data words, over the same alphabets, and conversely. A non-erasing production $(u, (v, o))$ can be encoded as the typed data word $\text{t2d}(u, (v, o)) = (v(1), u(d_1), d_1) \dots (v(n), u(d_k), d_k)$ where $d_i = o(i)$. The inverse of $\text{t2d}(u, (v, o))$ is defined, for all typed data words $w = (\gamma_1, \sigma_{d_1}, d_1) \dots (\gamma_k, \sigma_{d_k}, d_k)$, by the non-erasing production $\text{t2d}^{-1}(w) = (u, (v, o))$ such that $v = \gamma_1 \dots \gamma_k$, $o(i) = d_i$, and $u = \sigma_1 \dots \sigma_n$ where $n = \max_i d_i$. As an example, consider the following production and typed data word:



Proposition 18. *Non-erasing productions of $\mathcal{PR}(\Sigma, \Gamma)$ and typed data words of $\mathcal{TDW}(\Sigma, \Gamma)$ are in bijection by t2d .*

This encoding transfers to definability in \mathcal{L}_D and \mathcal{L}_T :

Theorem 19. *A non-erasing transduction τ is \mathcal{L}_T -definable iff $\text{t2d}(\tau)$ is \mathcal{L}_D -definable. Conversely, a language of typed data words L is \mathcal{L}_D -definable iff $\text{t2d}^{-1}(L)$ is \mathcal{L}_T -definable.*

Sketch of Proof. To go from \mathcal{L}_T to \mathcal{L}_D , the main idea is to make a syntactic transformation that mimics the encoding t2d : once inconsistent use of terms have been removed (such as e.g., $o(x) \leq_{\text{out}} y$), terms $o^n(x)$ are replaced by x , predicates \leq_{in} by \preceq and \leq_{out} by \leq . The converse is similar. \square

Thanks to Proposition 17, the satisfiability of \mathcal{L}_T reduces to satisfiability of \mathcal{L}_T over non-erasing transductions. Therefore, the combination of the theorems 16 and 19 will lead to the decidability of \mathcal{L}_T (Theorem 8). This result, together with other interesting results on \mathcal{L}_T , is proved in Section V.

IV. DECIDABILITY OF \mathcal{L}_D

This section is devoted to proving the decidability of the logic \mathcal{L}_D , and hereby of the logic \mathcal{L}_T . The proof scheme uses sets of labelled points, as done in [17], and unfolds as follows. We first get a normal form for the logic as formulas with quantifier depth at most 2 (Scott normal form). Then we observe that data words can be seen as two-dimensional structures with the horizontal axis representing the linear order and the vertical axis representing the order over data. Normalised \mathcal{L}_D -formulas are then translated into a set of constraints over sets of labelled points.

Then, the satisfiability test of these constraints is based on a bounded abstraction of horizontal lines (sets of labelled points aligned horizontally) called profiles. We define an automaton recognizing sequences of profiles which can be effectively turned into sequences of lines (thus forming a labelled point

system) satisfying the constraints. Hence, the decidability of an \mathcal{L}_D -formula reduces to testing emptiness of this profile automaton.

A. Scott normal form

The first step is to normalise any formula in \mathcal{L}_D into a Scott normal form (SNF). The procedure to put a formula in SNF is the same as FO^2 logics in general (see [23] for instance). We prove it in our context along with some preservation property about the data. Since we aim to get stronger properties than satisfiability, we state a stronger result, yet the proof is similar.

Lemma 20. *For any \mathcal{L}_D -formula φ over an alphabet Γ and a type alphabet Σ , one can construct an \mathcal{L}_D -formula ϕ over $\Gamma \times \Sigma$ and the type alphabet Σ such that:*

- *up to projection on Γ , ϕ and φ have the same models,*
- *ϕ is of the form $\forall x \forall y \psi(x, y) \wedge \bigwedge_{i=1}^m \forall x \exists y \psi_i(x, y)$ where the formulas ψ and ψ_i , $i = 1, \dots, m$, are quantifier free.*

Sketch of proof. The idea is to iteratively replace subformulas of quantifier depth one by a new predicate, and preserve the models by adding a new conjunct. At each step, a subformula $\exists y \psi(x, y)$ (resp. $\forall y \psi(x, y)$) where $\psi(x, y)$ is a quantifier free formula is replaced in φ by a new predicate $P(x)$. We add a conjunct $\forall x \exists y (P(x) \rightarrow \psi(x, y))$ (resp. $\forall x \forall y (P(x) \rightarrow \psi(x, y))$). The number of steps, and so the number of predicates added, is equal to the number of quantifications in φ . \square

B. 2-dimensional constraints over labelled point systems

Similarly to [17], we translate typed data words into sets of labelled points in a plane and positive formulas of \mathcal{L}_D of quantifier depth at most two into sets of constraints, while preserving satisfiability. The main differences with [17] are that each horizontal line will be additionally labelled by a data type, and that the constraints will use arbitrary MSO definable binary predicates interpreted on the word of line labels, instead of just the order and successor over lines.

Formally, we call *labelled point system* (LPS) over two disjoint alphabets Σ and Γ a pair of partial assignments of finite domains $L : \mathbb{N} \rightarrow \Sigma$ and $P : \mathbb{N}^2 \rightarrow \Gamma$ such that:

- 1) $\text{dom}(L)$ and $\text{dom}(P)$'s 1st-projection are downward-closed
- 2) $\text{dom}(P)$'s 2nd-projection equals $\text{dom}(L)$
- 3) no two points share the same abscissa, i.e. if $(i, j), (i, j') \in \text{dom}(P)$, then $j = j'$.

As we will see, the conditions 1 to 3 will ensure the existence of an isomorphism from LPS to typed data words. A *point* p of P is a triple (γ, i, j) such that $P(i, j) = \gamma$. A *line* is a set of points with same ordinate. The *line label* of a point $p = (\gamma, i, j) \in P$ is the label σ such that $L(j) = \sigma$. We denote by $\mathcal{LPS}(\Sigma, \Gamma)$ the set of LPS over Σ and Γ .

Thanks to the finiteness and downward-closedness of the domain of L , L is nothing but a word over Σ . As illustrated in Fig. 2, any LPS (L, P) can be viewed as a typed data word $\text{d2p}(L, P)$. Indeed, thanks to condition (3) and downward-closedness of P , there is exactly one point per abscissa up to some bound n . By condition (3), we have $P = \{(\gamma_1, 1, j_1), \dots, (\gamma_n, n, j_n)\}$, and we set $\text{d2p}(L, P) =$

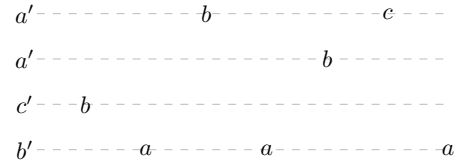


Fig. 2. An example of labelled point system (LPS). The corresponding typed data word is $(b, c', 2)(a, b', 1)(b, a', 4)(a, b', 1)(c, a', 3)(b, a', 4)(a, b', 1)$.

$(\gamma_1, L(j_1), j_1) \dots (\gamma_n, L(j_n), j_n)$ (note that thanks to condition (2), the set of data is downward-closed). Conversely, from any typed data word $w = (\gamma_1, \sigma_{d_1}, d_1) \dots (\gamma_n, \sigma_{d_n}, d_n)$ we get the LPS $\text{d2p}^{-1}(w) = (L, P)$ where $P = \{(\gamma_i, i, d_i) \mid 1 \leq i \leq n\}$ and $L = \{(\sigma_{d_i}, d_i \mid 1 \leq i \leq \max_i(d_i))\}$.

Proposition 21. *Typed data words of $\mathcal{TDW}(\Sigma, \Gamma)$ and labelled point systems of $\mathcal{LPS}(\Sigma, \Gamma)$ are in bijection by d2p .*

Constraints on LPS are built over label predicates, and some horizontal and vertical predicates. A label predicate $\gamma \in \Gamma$ is satisfied by a point p of P if p is labelled γ . Horizontal predicates are just horizontal directions \rightarrow, \leftarrow , which are satisfied by a pair of points $((\gamma, i, j), (\gamma', i', j'))$ if, respectively, $i < i'$ and $i' < i$. Vertical predicates are any MSO-definable binary predicate over line labels Σ and the vertical order, denoted by \leq . The pair of points $((\gamma, i, j), (\gamma', i', j'))$ satisfies a vertical predicate $\psi(x, y) \in \text{MSO}[\Sigma, \leq]$ if $(L, x = j, y = j') \models \psi(x, y)$.

An *existential constraint* is a pair (γ, E) where $\gamma \in \Gamma$ and E is a possibly empty set of tuples (γ', d, ψ) such that d is a horizontal direction and ψ is a vertical predicate. Given (L, P) an LPS, a point p of P satisfies an \exists -constraint (γ, E) if either p does not satisfy γ , or there exists a point q of P and a triple (γ', d, ψ) of E s.t. q satisfies γ' , and (p, q) satisfies d and ψ . In the latter case, we call q a *valid witness* of p for (γ, E) .

A *universal constraint* is a tuple $(\gamma, \gamma', d, \Psi)$ where d is a horizontal direction and Ψ a vertical predicate. A pair (p, q) of points of P satisfy a \forall -constraint $(\gamma, \gamma', d, \Psi)$ if p is not labelled by γ , q is not labelled by γ' , or (p, q) does not satisfy either d or Ψ . Then a universal constraint can be thought of as a forbidden pattern over pairs of points.

An instance of the (two-dimensional) MSO labelled point problem (MSOLPP) is a pair $C = (C_\exists, C_\forall)$ of sets of existential and universal constraints respectively. An LPS $M = (L, P)$ is a solution of (or model for) C , denoted $M \models C$, if every point of P satisfies all constraints in C_\exists and every pair of points satisfy all constraints in C_\forall .

C. From \mathcal{L}_D to MSOLPP

Proposition 22. *From any \mathcal{L}_D -sentence ϕ in SNF over Σ and Γ , one can construct an instance C of MSOLPP such that for any typed data word $w \in \mathcal{TDW}(\Sigma, \Gamma)$, $w \models \phi$ iff $\text{d2p}(w) \in \mathcal{LPS}(\Sigma, \Gamma)$ is a solution of C .*

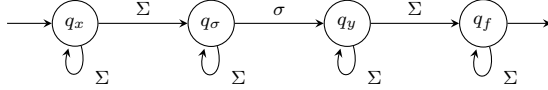
Sketch of proof. The key to the proof is that the conjunct $\forall x \forall y \psi(x, y)$ is translated into a universal constraint while

conjuncts of the form $\forall x \exists y \psi_i(x, y)$ become existential constraints. We consider atomic types for pairs of points, which are complete sets of truth values for all predicates (MSO, labels and directions). Then the conjunct $\forall x \forall y \psi(x, y)$ defines the set of types that are allowed (the types that satisfy $\psi(x, y)$), whose complement is then translated into universal constraints. Using simple formula manipulations, $\bigwedge_{i=1}^m \forall x \exists y \psi_i(x, y)$ is rewritten into $\forall x \bigwedge_{\gamma \in \Gamma} (\gamma(x) \rightarrow \exists y \bigvee_{\ell=1}^m t_{\gamma, \ell})$ where $t_{\gamma, \ell}$ is an atomic type. Then for each letter γ of Γ , we construct an existential constraint over some of the types $t_{\gamma, \ell}$. \square

D. Automata for binary predicates

It is well-known (see e.g. [24]) that any binary MSO $[\Sigma, \leq]$ -predicate $\psi(x, y)$ over Σ -labelled words, can be equivalently defined by a non-deterministic finite automaton (called here a predicate automaton) $\mathcal{A}_\psi = (Q_\psi, \Sigma, I_\psi, \Delta_\psi, F_\psi)$ equipped with a set $SP_\psi \subseteq Q_\psi^2$ of *selecting pairs* with the following semantics: for any word $u \in \Sigma^*$ and any pair of positions (i, j) of u , we have $u \models \psi(i, j)$ if, and only if, there exists an accepting run π of \mathcal{A}_ψ and a pair $(p, q) \in SP_\psi$ s.t. π is in state p before reading $u(i)$ and in state q before reading $u(j)$.

Example 23. Let us consider as an example the binary between predicate $Bet_\sigma(x, y) = \exists z \sigma(z) \wedge (x < z) \wedge (z < y)$, which cannot be expressed using only two variables. The automaton for this predicate is:



E. Profiles

In order to obtain the decidability of MSOLPP, we consider bounded abstractions of horizontal lines of labelled point systems. The abstraction of a horizontal line is called a *profile*. Intuitively, given an LPS, the profile of a line in this system holds information regarding some points on the line, but also regarding their relative position with respect to other points of the system that are relevant to the satisfiability of the constraints. A profile will be considered valid if for every existential constraint and every point, it describes a valid witness for it, and it never declares a pair of points which violates a universal constraint. Two successive profiles are consistent if they agree on the points described by each profile, meaning that if a point is declared in one profile, one can find a corresponding point in the other one. A sequence of profiles will be maximal if the profiles do not withhold information, which is relevant to the satisfiability of constraints.

1) *Profiles*: Let C be an instance of MSOLPP over Σ and Γ , and Ψ the set of MSO-predicates occurring in C . For all $\psi \in \Psi$, we let \mathcal{A}_ψ with set of states Q_ψ and set of selecting pairs SP_ψ be the predicate automaton for ψ . Let $S_\Psi = \biguplus_{\psi \in \Psi} Q_\psi$.

A C -*profile* (or just *profile*) is a tuple $\lambda = (\sigma, S, A_1 \dots A_n)$ where $\sigma \in \Sigma$ is a line label, $S \subseteq S_\Psi$ and $A_1 \dots A_n$ is a sequence of elements from $\Gamma \times (\{\cdot\} \cup \mathcal{P}(S \times S_\Psi) \times \{\uparrow, \downarrow\})$, called *clauses*, such that any clause A_k appears at most twice

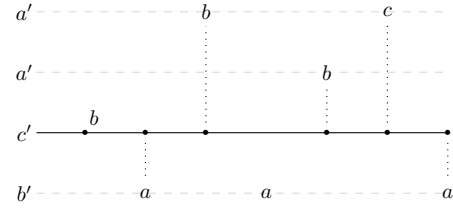


Fig. 3. A line with profile: $c', \{q_x, q_{a'}\}, (b, \cdot)(a, \begin{smallmatrix} (q_x, q_x) \\ (q_{a'}, q_x) \end{smallmatrix}, \downarrow) (b, \begin{smallmatrix} (q_x, q_y) \\ (q_{a'}, q_y) \end{smallmatrix}, \uparrow)(b, \begin{smallmatrix} (q_x, q_{a'}) \\ (q_{a'}, q_{a'}) \end{smallmatrix}, \uparrow)(c, \begin{smallmatrix} (q_x, q_y) \\ (q_{a'}, q_y) \end{smallmatrix}, \uparrow)(a, \begin{smallmatrix} (q_x, q_x) \\ (q_{a'}, q_x) \end{smallmatrix}, \downarrow)$

in the sequence, for all $k = 1, \dots, n$. By definition, the number of profiles is bounded by $N = |\Sigma| \cdot 2^{|S_\Psi|} (|\Gamma| \cdot (2(2^{|S_\Psi|} + 1) + 1))!$.

2) *Profile of a line*: Formal definitions can be found in Appendix. Given an LPS $M = (L, P)$ and an instance C of MSOLPP, we construct the profile of a line k of M as follows. The line label is the k th label of L , and the set of states S is the set of reachable and co-reachable states of the predicate automata of C before reaching position k of the vertical word. Then for each point p of P , we construct a clause of type (γ, \cdot) if p is on line k , or a clause containing the label of p , the set of all pairs of states such that there exists an accepting run that reaches the first state on k and the second on the line of p , and the direction (\uparrow or \downarrow) toward p . These clauses are ordered by the abscissa of the corresponding point, and we only keep the leftmost and rightmost occurrence of each clause.

Example 24. Let us give an example making use of line labels and which illustrates predicate automata. Figure 3 shows the relevant points for the profile of the second line of the LPS from Figure 2, when considering the binary predicate $Bet_{a'}$ as defined in Example 23. The complete profile sequence of the same LPS is given in the Appendix.

Notice that the point labelled by 'a' in the middle of the bottom line does not appear in any clause since it is not relevant to the satisfaction of constraints.

3) *Valid profiles*: A profile $\lambda = (\sigma, S, A_1 \dots A_n)$ satisfies an existential constraint $c = (\gamma, E)$ if for every i such that $A_i = (\gamma, \cdot)$ there exists a tuple (γ', d, ψ) of E , $j \leq n$ such that $i \sim_d j$, $(p, q) \in SP_\psi$ and $v \in \{\uparrow, \downarrow\}$ such that $(\gamma', \{(p, q)\}, v) \in A_j$. It satisfies a universal constraint $(\gamma, \gamma', d, \psi)$ if there does not exist i and j such that $i \sim_d j$, $A_i = (\gamma', \cdot)$ and $(\gamma', \{(p, q)\}, v) \in A_j$ for some $(p, q) \in SP_\psi$ and v a vertical direction. Given an instance C of MSOLPP, a profile is C -*valid* (or just *valid*) if it satisfies every constraint. A sequence of valid profiles is also called *valid* sequence.

4) *Consistency*: The states information contained in a profile declares, on a given line, every accepting run that can occur and influence the satisfaction of a constraint. When reading a sequence of profiles, we then have to ensure that what is declared by a profile is sound and complete. To this end, we define below the *consistency* of two consecutive profiles. The formal definition is technical and can be found in Appendix. Informally, two profiles are consistent if everything that is declared by one of the profiles is acknowledged by the other.

A clause A in a profile λ is matched by a clause A' in an other profile λ' if A and A' describe the same runs up to a single step. Then two profiles λ and λ' are *consistent* if we can order the clauses of both profiles in a way that is compatible with the respective orders of both profiles. Then, each clause of λ must be matched to a clause of λ' that is equal in this order, or appear inbetween two identical matching clauses of λ' .

A profile is *initial* (resp. *final*) if all the states of S are initial (resp. final). A sequence of profiles $\lambda_1 \dots \lambda_n$ is *consistent* if λ_1 is initial, λ_n is final, and for all $i < n$, λ_i is consistent with λ_{i+1} . It is *maximal* if it is consistent and one cannot add states in the sets S or in the clauses without making it inconsistent.

F. Satisfiability of MSOLPP

We now sketch the ideas for proving the following result:

Theorem 25. *MSOLPP satisfiability problem is decidable.*

The heart of the proof lies in the next two lemmas, which show that one only needs to consider valid, consistent and maximal sequences of profiles to check satisfiability of an MSOLPP. Given an LPS M , we denote by $\text{Seq}(M)$ the sequence of profiles of its horizontal lines (ordered from bottom to top). The next two lemmas show that given an instance C of MSOLPP, the set $\{\text{Seq}(M) \mid M \models C\}$ is exactly the set of valid, consistent and maximal sequences of C -profiles.

Lemma 26. *Given an instance C of MSOLPP, for any model M of C , $\text{Seq}(M)$ is C -valid, consistent and maximal.*

Sketch of proof. By definition of the profile of a line, all clauses are kept except for the ones that are not the leftmost nor rightmost copy. Hence when you consider two consecutive profiles, every clause is either matched or appears inbetween two identical matching clauses in the other profile, thus each pair of profiles is consistent. Since each state in the set of states of each profile is reachable and co-reachable, the bottom (resp. top) profile is initial (resp. final) and the sequence is consistent. Maximality comes from the definition of the profile of a line: each clause is constructed in a maximal way. Then if M is a model, any point has a witness, and a clause for the witness will appear in the profile of its line. Conversely, if two clauses violate a universal constraint, we can trace the clauses back to two points of M that violate it, concluding the proof. \square

Lemma 27. *Given an instance C of MSOLPP and a valid, consistent and maximal sequence of C -profiles $s = \lambda_1 \dots \lambda_n$, there exists a model M of C such that $s = \text{Seq}(M)$.*

Sketch of proof. The main difficulty is to place the clauses (γ, \cdot) in two-dimensional space. The idea is to use the consistency to order all the clauses of each pair of consecutive profiles. Combining the order of each pair of consecutive profiles, we get a pre-order over all clauses. By linearizing this pre-order and projecting it over the clauses (γ, \cdot) , we get a LPS M . It is then straightforward to check that $\text{Seq}(M) = s$ and that M is a model of C . \square

This final lemma states the regularity of sequences of profiles associated with models of MSOLPP instances.

Lemma 28. *Given an instance C of MSOLPP, the set $\{\text{Seq}(M) \mid M \models C\}$ is effectively regular.*

Sketch of Proof. By Lemmas 26 and 27, the set $\{\text{Seq}(M) \mid M \models C\}$ is exactly the set of profile sequences that are C -valid, consistent and maximal. Since these properties are local, one can construct a finite automaton over profiles that accept valid and consistent sequences of profiles. The non maximality is checked by a non-deterministic automaton that guesses a new accepting run that could be added to a sequence of clauses. Then the profile automaton is constructed by symmetric difference of these two automata. \square

Theorem 25 directly follows from Lemma 28: it suffices to check the emptiness of $\{\text{Seq}(M) \mid M \models C\}$.

Decidability of \mathcal{L}_D : Proof of Theorem 16. Let ψ be an \mathcal{L}_D -formula over Σ and Γ . We put ψ in SNF while preserving satisfiability (Lemma 20), and apply Proposition 22 to get an equivalent instance of MSOLPP C , whose satisfiability is in turn decidable by Theorem 25.

V. DECIDABILITY AND PROPERTIES OF \mathcal{L}_T

Decidability of \mathcal{L}_T : Proof of Theorem 8. Since by Proposition 17 any \mathcal{L}_T -transduction can be turned into a non-erasing one while preserving satisfiability, and since \mathcal{L}_T -definability of non-erasing transductions is equivalent to \mathcal{L}_D -definability through the encoding t2d (Theorem 19), we get the equivalence between Theorems 8 (\mathcal{L}_T decidability) and 16 (\mathcal{L}_D decidability). The latter has been proved in Section IV.

Other results. As a byproduct of the automata approach, we obtain interesting results on two-dimensional constraint systems that translate back to properties of the logic \mathcal{L}_T . Indeed, looking carefully, we observe that many properties of transductions are preserved all the way from \mathcal{L}_T formulas to MSOLPP and back. The properties of a transduction that we look at are the regularity of the input domain, functionality, origin boundedness and bounded origin uniformisation. A transduction with origin φ has *bounded origin* if there exists a $k \in \mathbb{N}$ s.t. any production $(u, (v, o)) \in \llbracket \varphi \rrbracket$ and any position i of u satisfies $|o^{-1}(i)| \leq k$. I.e. any position can only produce up to k letters. A transduction τ has a *bounded origin uniformisation* if there exists $k \in \mathbb{N}$ and a function τ' s.t. $\text{dom}(\tau) = \text{dom}(\tau')$, $\tau' \subseteq \tau$ and τ' has k -bounded origin. The next result settles these questions on LPS:

Theorem 29. *Let C be an instance of MSOLPP and $\llbracket C \rrbracket$ be its set of valid models. Then:*

- *The set of vertical words of $\llbracket C \rrbracket$ is regular.*
- *It is decidable if all models of $\llbracket C \rrbracket$ have bounded line width.*
- *It is decidable if there exist two different models with the same vertical word.*
- *There is $k \in \mathbb{N}$ such that to any vertical word of $\llbracket C \rrbracket$ one can associate a model of line width at most k .*

By translating this theorem back to transductions we get:

Corollary 30. *Given an \mathcal{L}_T -transduction τ :*

- *The domain of τ is regular.*
- *It is decidable whether τ has bounded origin.*
- *It is decidable whether τ is functional.*
- *τ has a bounded origin uniformisation.*

The last item means that for any \mathcal{L}_T -transduction, all valid inputs have an image of linearly bounded size. In particular, \mathcal{L}_T -definable functions are linear size increase.

VI. DISCUSSION AND FUTURE DIRECTIONS

We have defined a new logic for transductions with decidable satisfiability problem. There are many future directions that ought to be investigated. First, since the logic is based on MSO, it has non-elementary complexity. Supported by the fact that over data words, the logic $\text{FO}^2[\Gamma, \preceq, S_d, \leq]$ is EXPSpace-c, one may get much better complexity for \mathcal{L}_T if the binary predicates are given for instance by query automata.

We proved that \mathcal{L}_T can express functions definable in MSOT. Conversely, we conjecture that any \mathcal{L}_T -definable function is MSOT-definable. The former result has a nice consequence: the decidability of the model-checking problem for systems modeled by 2-way deterministic transducers (which are equivalent to MSOT) against specifications written in \mathcal{L}_T . Again, due to the expressiveness of MSO, the complexity is non-elementary and we believe that \mathcal{L}_T is a first step towards logics with much better model-checking complexities, which could be inspired by temporal logics. Related to that question is the definition of an automata model equivalent to \mathcal{L}_T . Automata have been defined for data words [20], [25], but none of these models seem to capture \mathcal{L}_T (modulo encoding). The automaton model from [25] for example is known to be equivalent to the undecidable logic $\exists\text{MSO}^2[S, S_d, \preceq]$ which we conjecture to be incomparable with \mathcal{L}_D (formula ϕ_{\leq}^a from Example 15 does not seem to be expressible in $\exists\text{MSO}^2[S, S_d, \preceq]$).

To get decidability, we had to break the expressiveness symmetry between input and output words, putting more power on the input part, to capture MSOT. Interesting questions are whether one could on the other hand have more expressive power on the output, while retaining decidability, at the price of weakening the expressiveness on the input. Obviously due to this asymmetry \mathcal{L}_T -transductions are not closed under inverse and we have shown that they are not closed under composition. It would be interesting to obtain an expressive and decidable logic closed under these properties.

Another direction is to extend the logic to other structures (e.g. trees and infinite words), and other predicates over output positions. However, one has to be careful since the data point of view shows that we are already close to undecidability (e.g. over data words, FO^2 with successor over data and positions is undecidable [26]). We will first investigate the extensions with modular predicates [27] and in-between predicates [28].

We established a tight connection between transductions and data words, and a new logic for data words. The data point of view allowed us to get decidability of the transduction logic

\mathcal{L}_T , inspired by the decidability result of [17]. On the other hand, we would like to investigate if there are results from the theory of transductions that would translate into interesting results in the theory of data words.

As a consequence of \mathcal{L}_T decidability, the equivalence problem with origin is decidable for \mathcal{L}_T -transductions. It is not difficult (and not surprising) to show that the equivalence problem of \mathcal{L}_T -transductions up to origin projection is undecidable (the undecidability of equivalence for finite-state transducers of [29] can be easily adapted). An interesting continuation would be to consider some relaxation of the equivalence problem, by comparing transductions with *similar* origin, as done for instance in [30] for rational relations.

REFERENCES

- [1] H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity*. Boston, Basel and Berlin: Birkhäuser, 1994.
- [2] V. Diekert and P. Gastin, “First-order definable languages,” in *Logic and Automata: History and Perspectives*, ser. Texts in Logic and Games, vol. 2. Amsterdam University Press, 2008, pp. 261–306.
- [3] H. Comon-Lundh, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, *Tree Automata Techniques and Applications*, 2007, (online) <http://www.grappa.univ-lille3.fr/tata/>.
- [4] W. Thomas, “Languages, automata and logic,” in *Handbook of Formal Languages*. Springer, 1997, vol. 3, Beyond Words.
- [5] M. Y. Vardi and P. Wolper, “An automata-theoretic approach to automatic program verification,” in *lics86*, 1986, pp. 332–344.
- [6] J. Berstel, *Transductions and Context-Free Languages*. Teubner, 1979.
- [7] J. Sakarovich, *Elements of Automata Theory*. Cambridge, England: Cambridge University Press, 2009.
- [8] E. M. Gurari, “The equivalence problem for deterministic two-way sequential transducers is decidable,” *SIAM JComp.*, 11(3):448–452, 1982.
- [9] M. P. Chytil and V. Ják, “Serial composition of 2-way finite-state transducers and simple programs on strings,” in *ALP*. Springer, 1977, pp. 135–137. LNCS, Vol. 52.
- [10] R. Alur and P. Černý, “Expressiveness of streaming string transducers,” in *FSTTCS*, ser. LIPIcs. Schloss Dagstuhl., 2010, vol. 8:1–12.
- [11] M. P. Schützenberger, “A remark on finite transducers,” *Information and Control*, vol. 4, no. 2-3, pp. 185–196, 1961.
- [12] E. Filiot, O. Gauwin, and N. Lhote, “First-order definability of rational transductions: An algebraic approach,” in *LICS*, 2016, pp. 387–396.
- [13] J. Engelfriet and H. J. Hoogeboom, “MSO definable string transductions and two-way finite-state transducers,” *ACM Trans. Comput. Log.*, vol. 2, no. 2, pp. 216–254, 2001.
- [14] B. Courcelle, “Monadic second-order definable graph transductions: a survey,” *TCS*, vol. 126, no. 1, pp. 53–75, 1994.
- [15] R. Alur and J. V. Deshmukh, “Nondeterministic streaming string transducers,” in *ICALP*, 2011, pp. 1–20.
- [16] M. Bojanczyk, “Transducers with origin information,” in *ICALP*, 2014.
- [17] T. Schwentick and T. Zeume, “Two-variable logic with two order relations,” *LMCS*, vol. 8, no. 1, 2012.
- [18] E. Filiot, S. Maneth, P. Reynier, and J. Talbot, “Decision problems of tree transducers with origin,” in *ICALP*, 2015, pp. 209–221.
- [19] L. Segoufin, “Automata and logics for words and trees over an infinite alphabet,” in *CSL*, 2006, pp. 41–57.
- [20] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David, “Two-variable logic on words with data,” in *LICS*, 2006, pp. 7–16.
- [21] B. Courcelle and J. Engelfriet, “Book: Graph structure and monadic second-order logic. A language-theoretic approach,” *Bulletin of the EATCS*, vol. 108, p. 179, 2012.
- [22] E. Filiot, “Logic-automata connections for transformations,” in *ICLA*, ser. LNCS, vol. 8923. Springer, 2015, pp. 30–57.
- [23] E. Grädel and M. Otto, “On logics with two variables,” *TCS*, vol. 224, no. 1-2, pp. 73–113, 1999.
- [24] J. Niehren, L. Planque, J. Talbot, and S. Tison, “N-ary queries by tree automata,” in *DBPL*, 2005, pp. 232–246.
- [25] A. Manuel and T. Zeume, “Two-variable logic on 2-dimensional structures,” in *CSL*, 2013, pp. 484–499.

- [26] A. Manuel, T. Schwentick, and T. Zeume, “A short note on two-variable logic with a linear order successor and a preorder successor,” *CoRR*, vol. abs/1306.3418, 2013.
- [27] L. Dartois and C. Paperman, “Two-variable first order logic with modular predicates over words,” in *STACS*, 2013, pp. 329–340.
- [28] A. Krebs, K. Lodaya, P. K. Pandya, and H. Straubing, “Two-variable logic with a between relation,” in *LICS*. ACM, 2016, pp. 106–115.
- [29] T. V. Griffiths, “The unsolvability of the equivalence problem for lambda-free nondeterministic generalized machines,” *JACM*, vol. 15, no. 3, pp. 409–413, 1968.
- [30] E. Filiot, I. Jecker, C. Löding, and S. Winter, “On equivalence and uniformisation problems for finite transducers,” in *ICALP*, ser. LIPIcs, vol. 55. Schloss Dagstuhl, 2016, pp. 125:1–125:14.

APPENDIX A FO AND MSO LOGICS

Example 2. First the true \top formula is satisfied by any production. Hence $\llbracket \top \rrbracket = \Sigma^+ \times \Gamma^*$. Let us now define several macros that will be useful throughout the paper. The formula $\text{in}(x) \equiv x \leq_{\text{in}} x$ (resp. $\text{out}(x) \equiv x \leq_{\text{out}} x$) hold true if x belongs to the input word (resp. output word).

For $\alpha \in \{\text{in}, \text{out}\}$, we define the guarded quantifiers $\exists^\alpha x \phi$ and $\forall^\alpha x \phi$ as shortcuts for $\exists x \alpha(x) \wedge \phi$ and $\forall x \alpha(x) \rightarrow \phi$ (note that $\neg \exists^\alpha x \phi$ is equivalent to $\forall^\alpha x \neg \phi$). Even if it is not necessary, we will often use guarded quantifiers for the sake of formula readability. Then, for $\alpha \in \{\text{in}, \text{out}\}$, the formula

$$S_\alpha(x, y) \equiv x <_\alpha y \wedge \forall^\alpha z \neg(x <_\alpha z \wedge z <_\alpha y)$$

defines the successor relations over the input and output positions. Then, the following formulas define the first and last position of the input and output words respectively:

$$\begin{aligned} \min_\alpha(x) &\equiv \alpha(x) \wedge \forall^\alpha y x \leq_\alpha y \\ \max_\alpha(x) &\equiv \alpha(x) \wedge \forall^\alpha y y \leq_\alpha x \end{aligned}$$

The following formulas express that the origin mapping is respectively an injection, surjection, bijection from output positions to input positions:

$$\begin{aligned} \phi_{\text{inj}} &\equiv \forall^{\text{out}} x, y (\mathbf{o}(x) = \mathbf{o}(y)) \rightarrow x = y \\ \phi_{\text{surj}} &\equiv \forall^{\text{in}} x \exists^{\text{out}} y x = \mathbf{o}(y) \\ \phi_{\text{bij}} &\equiv \phi_{\text{inj}} \wedge \phi_{\text{surj}} \end{aligned}$$

We now have all necessary ingredients to define the transduction τ_{shuffle} of Example 1:

$$\phi_{\text{shuffle}} \equiv \phi_{\text{bij}} \wedge \forall^{\text{out}} x \bigwedge_{\sigma \in \Sigma} \sigma(\mathbf{o}(x)) \rightarrow \bar{\sigma}(x)$$

If we additionally ask that the origin mapping preserves the order, we get a formula which defines the identity transduction f_{id} of Example 1:

$$\begin{aligned} \phi_{\text{order-pres.}} &\equiv \forall^{\text{out}} x \forall^{\text{out}} y x \leq_{\text{out}} y \rightarrow \mathbf{o}(x) \leq_{\text{in}} \mathbf{o}(y) \\ \phi_{\text{id}} &\equiv \phi_{\text{shuffle}} \wedge \phi_{\text{order-pres.}} \end{aligned}$$

Formulas of Example 2 as \mathcal{L}_T -formulas. Let us consider again all the formulas of Example 2. Modulo putting brackets, they are all in \mathcal{L}_T :

$$\begin{aligned} \text{in}(x) &\equiv \{x \leq_{\text{in}} x\} & \text{out}(x) &\equiv x \leq_{\text{out}} x \\ S_{\text{in}}(x, y) &\equiv \{x <_{\text{in}} y \wedge \forall^{\text{in}} z \neg(x <_{\text{in}} z \wedge z <_{\text{in}} y)\} \\ S_{\text{out}}(x, y) &\equiv x <_{\text{out}} y \wedge \forall^{\text{out}} z \neg(x <_{\text{out}} z \wedge z <_{\text{out}} y) \\ \min_{\text{in}}(x) &\equiv \text{in}(x) \wedge \{\forall^{\text{in}} y x \leq_{\text{in}} y\} \\ \max_{\text{in}}(x) &\equiv \text{in}(x) \wedge \{\forall^{\text{in}} y y \leq_{\text{in}} x\} \\ \min_{\text{out}}(x) &\equiv \text{out}(x) \wedge \forall^{\text{out}} y x \leq_{\text{out}} y \\ \max_{\text{out}}(x) &\equiv \text{out}(x) \wedge \forall^{\text{out}} y y \leq_{\text{out}} x \\ \phi_{\text{inj}} &\equiv \forall^{\text{out}} x \forall^{\text{out}} y \{\mathbf{o}(x) = \mathbf{o}(y)\} \rightarrow x = y \\ \phi_{\text{surj}} &\equiv \forall^{\text{in}} x \exists^{\text{out}} y \{x = \mathbf{o}(y)\} \\ \phi_{\text{bij}} &\equiv \phi_{\text{inj}} \wedge \phi_{\text{surj}} \\ \phi_{\text{shuffle}} &\equiv \phi_{\text{bij}} \wedge \forall^{\text{out}} x \bigwedge_{\sigma \in \Sigma} \{\sigma(\mathbf{o}(x))\} \rightarrow \bar{\sigma}(x) \\ \phi_{\text{id}} &\equiv \phi_{\text{shuffle}} \wedge \phi_{\text{order-pres.}} \end{aligned}$$

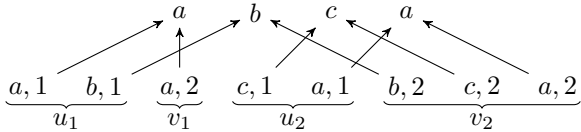
APPENDIX B
SECTION II-C: UNDECIDABILITY OF
 $\text{FO}^2[\Sigma, \Gamma, \leq_{\text{IN}}, \leq_{\text{OUT}}, S_{\text{OUT}}, \mathbf{O}]$

Proposition 5. *Over transductions, the logic $\text{FO}^2[\Sigma, \Gamma, \leq_{\text{IN}}, \leq_{\text{OUT}}, S_{\text{OUT}}, \mathbf{O}]$ is undecidable.*

Proof. The proof is a reduction from the Post Correspondence Problem (PCP) and is an adaptation in the context of transductions of an undecidability proof for the two-variable fragment of first-order logic for data words with a linear-order \leq and an equivalence relation \sim [20].

Given an alphabet A and n pairs $(u_i, v_i) \in A^+ \times A^+$ (they can be assumed to be non-empty without losing undecidability), we construct a sentence $\phi \in \text{FO}[\mathcal{T}_{\Sigma, \Gamma}]$ which is satisfiable iff there exist $i_1, \dots, i_k \in \{1, \dots, n\}$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$. We let $\Sigma = A$ and $\Gamma = A_1 \cup A_2$, where $A_i = A \times \{i\}$. Given a word $w = a_1 \dots a_p \in A^*$ and $\ell = 1, 2$, we let $\ell(w) = (a_1, \ell) \dots (a_p, \ell) \in A_\ell^*$. For any two sequences of words $s = w_1, w_2, \dots, w_k \in A^*$ and $s' = w'_1, \dots, w'_k \in A^*$, we define $s \otimes s' \in \Gamma^*$ their interleaving, by $1(w_1)2(w'_1)1(w_2)2(w'_2) \dots 1(w_k)2(w'_k)$. E.g. $(ab, ca) \otimes (a, bca) = (a, 1)(b, 1)(a, 2)(c, 1)(a, 1)(b, 2)(c, 2)(a, 2)$.

We will construct the formula ϕ in such a way that it defines the transduction from Σ to Γ which maps any word $u \in \Sigma^*$ for which there exist $i_1, \dots, i_k \in \{1, \dots, n\}$ such that $u_{i_1} \dots u_{i_k} = u = v_{i_1} \dots v_{i_k}$, to $w = (u_{i_1}, \dots, u_{i_k}) \otimes (v_{i_1}, \dots, v_{i_k})$, with origin mapping o which maps any position of w corresponding to some u_{i_j} (or to some v_{i_j}) to the same position in u . E.g., over $A = \{a, b, c\}$, if one takes $u_1 = ab$, $u_2 = ca$, $v_1 = a$, $v_2 = bca$, then the sequence 1, 2 is a solution to PCP, and it gives rise to the following production:



First, we express that the output word is of the form $(u_{i_1}, \dots, u_{i_k}) \otimes (v_{i_1}, \dots, v_{i_k})$ for some i_1, \dots, i_k . For that, we need to define a formula $\phi_{\text{cut}}(x)$ which holds true at output position x if either x is the first output position, or it is labelled in A_1 while its predecessor is labelled in A_2 :

$$\phi_{\text{cut}}(x) \equiv \forall^{\text{out}} y \cdot S_{\text{out}}(y, x) \rightarrow A_1(x) \wedge A_2(y)$$

where for all $\ell = 1, 2$, $A_\ell(x)$ stands for $\bigvee_{a \in A} (a, \ell)(x)$.

Now, the idea when x is a cut (i.e. satisfies the formula $\phi_{\text{cut}}(x)$), is to guess an index $i \in \{1, \dots, n\}$ and check that the sequence of labels from position x (x included) to the next cut (if it exists) or to the end (if not) is $1(u_i)2(v_i)$.

To define this, we introduce, for all formulas ϕ with one free variable, the formula $\zeta\phi_j(x)$ which holds true if the j -th successor of x exists and satisfies ϕ . It is inductively defined by:

$$\zeta\phi_0(x) \equiv \phi(x) \quad \zeta\phi_j(x) \equiv \exists^{\text{out}} y \cdot S_{\text{out}}(x, y) \wedge \zeta\phi_{j-1}(y)$$

where y is a variable different from x . Then, we define the following formula for $i \in \{1, \dots, n\}$:

$$\begin{aligned} \phi_{u_i, v_i}(x) \equiv & \bigwedge_{j=1}^{|u_i|} \zeta(u_i(j), 1)(x) \zeta_{j-1}(x) \\ & \wedge \bigwedge_{j=1}^{|v_i|} \zeta(v_i(j), 2)(x) \zeta_{j-1+|u_i|}(x) \\ & \wedge \zeta_{\text{cut}}(x) \vee \text{max}_{\text{out}}(x) \zeta_{|u_i|+|v_i|-1}(x) \end{aligned}$$

Finally, the following formula expresses that the output word is of the form $(u_{i_1}, \dots, u_{i_k}) \otimes (v_{i_1}, \dots, v_{i_k})$ for some i_1, \dots, i_k :

$$\phi_{\text{well-formed}} \equiv \forall^{\text{out}} x \cdot (\phi_{\text{cut}}(x) \rightarrow \bigvee_{i=1}^n \phi_{u_i, v_i}(x))$$

So far, we have not checked any property of the origin mapping, nor the fact that the output decomposition satisfies $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k} = u$ if u is the input word. To achieve that, it remains to express, for all $\ell = 1, 2$, that the origin mapping restricted to positions labelled in A_ℓ is bijective and preserves the orders and labels.

$$\begin{aligned} \phi_{\text{bij}, \ell} &\equiv \forall^{\text{in}} x \exists^{\text{out}} y \ A_\ell(y) \wedge \mathbf{o}(y) = x \\ &\quad \wedge \forall^{\text{out}} x, y \ (\mathbf{o}(x) = \mathbf{o}(y) \wedge A_\ell(x) \wedge A_\ell(y)) \\ &\quad \rightarrow x = y \\ \phi_{\text{ord-pres}, \ell} &\equiv \forall^{\text{out}} x, y \ (\mathbf{o}(x) <_{\text{in}} \mathbf{o}(y) \wedge A_\ell(x) \wedge A_\ell(y)) \\ &\quad \rightarrow x <_{\text{out}} y \\ \phi_{\text{lab-pres}, \ell} &\equiv \forall^{\text{out}} x \bigwedge_{a \in A} (a, \ell)(x) \rightarrow a(\mathbf{o}(x)) \end{aligned}$$

The final formula ϕ is then:

$$\phi \equiv \phi_{\text{well-formed}} \wedge \bigwedge_{\ell=1}^2 \phi_{\text{bij}, \ell} \wedge \phi_{\text{ord-pres}, \ell} \wedge \phi_{\text{lab-pres}, \ell}$$

Note that we have only used two variables x and y all over the construction. \square

APPENDIX C
SECTION II-D

Corollary 10. *Over transductions, the logic $\exists \text{MSO}^2[\Gamma, \leq_{\text{out}}, \mathbf{O}, \text{MSO}_{\text{bin}}[\leq_{\text{in}}, \Sigma]]$ is decidable.*

Proof. To test the satisfiability of a formula $\exists X_1 \dots \exists X_n \phi$ where ϕ is in $\text{FO}^2[\Gamma, \leq_{\text{out}}, \mathbf{O}, \text{MSO}_{\text{bin}}[\leq_{\text{in}}, \Sigma], X_1, \dots, X_n]$, it suffices to encode the predicates X_1, \dots, X_n into a new alphabet, in order to get an \mathcal{L}_T -formula. More precisely, we define $\Gamma' = \Gamma \times 2^{\{X_1, \dots, X_n\}}$ and $\Sigma' = \Sigma \times 2^{\{X_1, \dots, X_n\}}$, and modify ϕ any atom $X_i(x)$ by

$$\bigvee_{(\gamma, P) \in \Gamma', X_i \in P} (\gamma, P)(x) \vee \bigvee_{(\sigma, P) \in \Sigma', X_i \in P} (\sigma, P)(x),$$

any atom $\gamma(x)$ by $\bigvee_{(\gamma, P) \in \Gamma'} (\gamma, P)(x)$, and any atom $\sigma(x)$ by $\bigvee_{(\sigma, P) \in \Sigma'} (\sigma, P)(x)$. Thus we get an \mathcal{L}_T -formula with input alphabet Σ' and output alphabet Γ' which is satisfiable iff $\exists X_1 \dots \exists X_n \phi$ is. \square

APPENDIX D

SECTION II-E: EXPRESSIVENESS OF \mathcal{L}_T

Theorem 13. Any MSO-transduction is \mathcal{L}_T -definable.

Proof of Theorem 13. Let ϕ be an MSO $[\Sigma, \leq]$ -formula, we denote by $\bar{\phi}$ the same formula in which the predicate \leq is replaced by \leq_{in} . First let us define some unary and binary predicates for the input. Let P be a subset of $\{1, \dots, k\}$, we define the formula which states that the copies of x which are used for the output are the ones of P :

$$\phi_P(x) = \bigwedge_{c \in P} \phi_{\text{pos}}^c(x) \bigwedge_{c \notin P} \neg \phi_{\text{pos}}^c(x)$$

Let c_1, \dots, c_l be a sequence of non-repeating integers smaller than k , then we define the formula which says that the order of the copies of x in the output follow the sequence:

$$\phi_{c_1, \dots, c_l}(x) = \phi_{\{c_1, \dots, c_l\}}(x) \bigwedge_{1 \leq i \leq j \leq l} (\phi_{\leq}^{c_i, c_j}(x, x))$$

Now let $v \in \Gamma^l$, we define the formula specifying the letters of the output positions:

$$\phi_{c_1, \dots, c_l, v}(x) = \phi_{c_1, \dots, c_l}(x) \bigwedge_{i \leq l} \phi_{v[i]}^{c_i}(x)$$

Let d_1, \dots, d_m be a sequence of non-repeating integers smaller than k and $w \in \Gamma^m$, then we define:

$$\phi_{c_1, \dots, c_l, v, d_1, \dots, d_m, w}(x, y) = \phi_{c_1, \dots, c_l, v}(x) \wedge \phi_{d_1, \dots, d_m, w}(y)$$

Now we define an \mathcal{L}_T -formula $C_i(x)$ which states that x is exactly the i th output position of some input position.

$$C_1(x) = \text{out}(x) \wedge \forall^{\text{out}} y \ y <_{\text{out}} x \rightarrow \{\text{o}(x) \neq \text{o}(y)\}$$

And for $i > 0$:

$$\begin{aligned} C_{i+1}(x) = & \exists^{\text{out}} y \ (y <_{\text{out}} x \wedge \{\text{o}(x) = \text{o}(y)\} \wedge C_i(y)) \\ & \wedge \forall^{\text{out}} y \ (y <_{\text{out}} x \wedge \{\text{o}(x) = \text{o}(y)\} \wedge C_i(y)) \\ & \rightarrow \neg \exists^{\text{out}} x \ (x <_{\text{out}} y \wedge \{\text{o}(x) = \text{o}(y)\} \wedge C_i(x)) \end{aligned}$$

Note that we have used only two variables x and y . Now we can define an \mathcal{L}_T formula which defines the MSO-transduction:

$$\begin{aligned} & \{\overline{\phi_{\text{dom}}}\} \wedge \forall^{\text{out}} x \neg C_{k+1}(x) \wedge \forall^{\text{in}} x \{\overline{\phi_{\emptyset}}\}(x) \rightarrow (\forall^{\text{out}} y \ \{\text{o}(y) \neq \text{o}(x)\} \\ & \wedge \forall^{\text{out}} x, y \ \bigwedge_{m, l \leq k, c_1, \dots, c_l, v \in \Gamma^l, d_1, \dots, d_m, w \in \Gamma^m, i \leq l, j \leq m} \\ & \quad (\overline{C_i(x)} \wedge C_j(y) \wedge \\ & \quad \quad \{\overline{\phi_{\leq}^{c_i, d_j}}\}(\text{o}(x), \text{o}(y)) \wedge \\ & \quad \quad \{\overline{\phi_{c_1, \dots, c_l, v, d_1, \dots, d_m, w}}\}(\text{o}(x), \text{o}(y))) \\ & \rightarrow (x \leq_{\text{out}} y \wedge v[i](x) \wedge w[j](y)) \end{aligned}$$

APPENDIX E

SECTION III-B: EXAMPLES OF \mathcal{L}_D FORMULAS

We give a translation of formulas from Example 6. We have already seen how to define the formulas ψ_{bij} and $\psi_{\text{order-pres.}}$.

A. Shuffle of a regular language

Let Γ be a (finite) alphabet. Given a language $L \subseteq \Gamma^*$, we define its *closure under shuffle* $\text{shuffle}(L)$ to be the set of words $\sigma_{\pi(1)} \dots \sigma_{\pi(n)}$ such that π is a bijection from $\{1, \dots, n\}$ to $\{1, \dots, n\}$, and $\gamma_1 \dots \gamma_n \in L$. If L is regular, one can define an \mathcal{L}_D -formula $\psi_{\text{shuffle}(L)}$ over Γ and some set of types Σ , such that $\gamma_1 \dots \gamma_n \in \text{shuffle}(L)$ iff there exists a typed data word $(\gamma_1, \sigma_1, d_1) \dots (\gamma_n, \sigma_n, d_n) \models \psi_{\text{shuffle}(L)}$.

We take $\Sigma = \{\bar{\gamma} \mid \gamma \in \Gamma\}$ (remind that we require Σ and Γ to be disjoint). For $L \subseteq \Gamma^*$, denote $\bar{L} \subseteq \Sigma^*$ its “bar annotated” version. If L is regular, then the set of typed data words $(\gamma_1, \sigma_1, d_1) \dots (\gamma_n, \sigma_n, d_n)$ such that $\sigma_{\pi(1)} \dots \sigma_{\pi(n)} \in \bar{L}$, for π a bijection from $\{1, \dots, n\}$ into $\{1, \dots, n\}$ such that $d_{\pi(1)} \leq \dots \leq d_{\pi(n)}$ is definable by an MSO $_{\text{bin}}[\Sigma, \leq]$ -predicate $\psi_{\bar{L}}$. Then, the formula $\psi_{\text{shuffle}(L)}$ is:

$$\psi_{\text{shuffle}(L)} \equiv \psi_{\text{bij}} \wedge \{\psi_{\bar{L}}\} \wedge \forall x \bigwedge_{\gamma \in \Gamma} \gamma(x) \rightarrow \{\bar{\gamma}(x)\}$$

B. Dyck language

Let $\Gamma = \{[,]\}$ and $\Sigma = \{a, b\}$. We define an \mathcal{L}_D -formula ψ_{Dyck} such that for all $\gamma_1 \dots \gamma_n \in \Gamma^*$, $\gamma_1 \dots \gamma_n$ is well-parenthesised iff there exists a typed data word $(\gamma_1, \sigma_1, d_1) \dots (\gamma_n, \sigma_n, d_n) \models \psi_{\text{Dyck}}$.

First, one expresses that the data (taken in order) alternate between type a and type b . Since it is a regular property, it is definable by some MSO $_{\text{bin}}[\Sigma, \leq]$ -predicate $\psi_{(ab)^*}$ defined by the conjunction of the following constraints:

$$\begin{aligned} & \forall x \forall y \ (a(x) \wedge S_{\leq}(x, y)) \rightarrow b(y) \\ & \forall x \forall y \ (b(x) \wedge S_{\leq}(x, y)) \rightarrow a(y) \\ & \forall x \ (\min_{\leq}(x) \rightarrow a(x)) \\ & \forall x \ (\max_{\leq}(x) \rightarrow b(x)) \end{aligned}$$

where $S_{\leq}(x, y) \equiv x \leq y \wedge x \not\sim y \wedge \neg \exists z \ (x \leq z \leq y \wedge x \not\sim z \wedge z \not\sim x)$, $\min_{\leq}(x) \equiv \forall y \ x \leq y$, $\max_{\leq}(x) \equiv \forall y \ y \leq x$.

Then, similarly to the same example for transductions, we define ψ_{Dyck} as follows:

$$\begin{aligned} \psi_{\text{lab}} & \equiv \forall x \ [(x) \rightarrow \{a(x)\} \wedge](x) \rightarrow \{b(x)\} \\ \psi_d & \equiv \forall x \forall y \ \{S_{\leq}(x, y) \wedge b(y)\} \rightarrow x \leq y \\ \psi_{\text{Dyck}} & \equiv \{\psi_{(ab)^*}\} \wedge \psi_{\text{bij}} \wedge \psi_{\text{lab}} \wedge \psi_d \end{aligned}$$

APPENDIX F

SECTION III-C: FROM TRANSDUCTIONS TO DATA WORDS

Proposition 17. For any \mathcal{L}_T -formula φ there exists a non-erasing \mathcal{L}_T -formula φ' such that $\text{dom}(\llbracket \varphi \rrbracket) = \text{dom}(\llbracket \varphi' \rrbracket)$. In particular, φ is satisfiable if, and only if, φ' is.

Proof. let φ be an \mathcal{L}_T -formula, we want to obtain an \mathcal{L}_T -formula $\varphi^{\text{n.e.}}$ which is non-erasing. The idea is to extend the output of all productions by a copy of the input word. We add a new output letter \sharp which will separate the normal output and the copy of the input. We want to obtain $(u, (v, o)) \models \varphi$ iff $(u, (v\sharp u, o')) \models \varphi^{\text{n.e.}}$ where $o'(i) = o(i)$ if $i \leq |v|$, $o'(1 + i + |v|) = i$ if $i \leq |u|$ and $o'(|v| + 1) = 1$. From φ , we construct $\varphi^{<\sharp}$ where every quantification over the output positions is relativised as being before a position labelled

by \sharp . Similarly, for ϕ_{id} the identity transduction, we define $\phi_{\text{id}}^{>\sharp}$ where quantifications over the output are relativised as appearing after a position labelled by \sharp . Adding the guards can be done while staying in the two-variable fragment. Then we define $\varphi^{\text{n.e.}}$ to be equal to:

$$\varphi^{<\sharp} \wedge \phi_{\text{id}}^{>\sharp} \wedge \exists^{\text{out}} x \sharp(x) \wedge \min_{\text{in}}(\mathbf{o}(x)) \wedge \forall^{\text{out}} y \sharp(y) \rightarrow x = y$$

□

Theorem 19. *A non-erasing transduction τ is \mathcal{L}_T -definable iff $\text{t2d}(\tau)$ is \mathcal{L}_D -definable. Conversely, a language of typed data words L is \mathcal{L}_D -definable iff $\text{t2d}^{-1}(L)$ is \mathcal{L}_T -definable.*

Proof. Let φ be an \mathcal{L}_T -sentence defining a non-erasing transduction, we want to obtain an \mathcal{L}_D -sentence ϕ defining its encoding as a typed data word. First we transform φ into φ' a formula where all quantifications are either input or output quantifications. This can be done inductively on \mathcal{L}_T -formula by replacing $\exists x F(x)$ by $\exists^{\text{in}} x F(x) \vee \exists^{\text{out}} x F(x)$. Then, we simplify the resulting formula by removing inconsistent use of variables in the predicates with respect to the type of their quantifiers. For that, we say that the occurrence of a term t is of type **in** if it is equal to x where x is quantified over the input, or of the form $\mathbf{o}(t')$ for some term t' . It is of type **out** if $t = x$ for x a variable quantified over the output. Now, we replace in ϕ all occurrences of the following atoms by \perp under the following conditions:

- the atom is $\gamma(t)$ and t is not of type **out**,
- the atom is $t_1 \leq_{\text{out}} t_2$ and some t_i is not of type **out**,
- the atom is $\{\psi\}(t_1, t_2)$ and some t_i is not of type **in**.

By doing this we obtain a new formula which is equivalent to ϕ , and makes a consistent use of its variables. We do not give a name to this new formula and rather assume that ϕ satisfies this property.

Then, we do the following replacement in ϕ to transform it into an equivalent \mathcal{L}_D -formula. First, similarly to the bijection from Prop. 18 in which the origin of a position becomes its data value, any term of the form $\mathbf{o}^n(x)$ is replaced by x . Then, any occurrence of an MSO predicate $\{\psi\}(x, y)$ is replaced by $\{\psi'\}(x, y)$, where ψ' is obtained by replacing in ψ all atoms of the form $x \leq_{\text{in}} y$ by $x \preceq y$. We also replace the atom of the form $x \leq_{\text{out}} y$ by $x \leq y$. If denote by ϕ' the obtained formula, by construction we have $(u, (v, o)) \models \phi$ iff $\text{t2d}(u, (v, o)) \models \phi'$.

Example 3. For instance, consider the following formula ϕ :

$$\forall x \{\sigma(x')\}(x) \rightarrow \exists y \{y' \leq_{\text{in}} x'\}(\mathbf{o}(y), x)$$

where $\sigma \in \Sigma$. It expresses the fact for any input position labelled σ , there is another input position before which is the origin of some output position. First, note that $\forall x \psi$ being a shortcut for $\neg \exists x \neg \psi$, the first replacement by typed quantifiers

gives the formula $\forall^{\text{in}} x \psi \wedge \forall^{\text{out}} x \psi$. Then, the first rewriting step of ϕ gives:

$$\begin{aligned} \forall^{\text{in}} x \{\sigma(x')\}(x) &\rightarrow \exists^{\text{in}} y \{y' \leq_{\text{in}} x'\}(\mathbf{o}(y), x) \\ &\quad \vee \\ &\quad \exists^{\text{out}} y \{y' \leq_{\text{in}} x'\}(\mathbf{o}(y), x) \\ \forall^{\text{out}} x \{\sigma(x')\}(x) &\rightarrow \exists^{\text{in}} y \{y' \leq_{\text{in}} x'\}(\mathbf{o}(y), x) \\ &\quad \vee \\ &\quad \exists^{\text{out}} y \{y' \leq_{\text{in}} x'\}(\mathbf{o}(y), x) \end{aligned}$$

After the simplification step according to types, we get:

$$\begin{aligned} \forall^{\text{in}} x \{\sigma(x')\}(x) &\rightarrow \exists^{\text{in}} y \perp \\ &\quad \vee \\ &\quad \exists^{\text{out}} y \{y' \leq_{\text{in}} x'\}(\mathbf{o}(y), x) \\ \forall^{\text{out}} x \perp &\rightarrow \exists^{\text{in}} y \perp \\ &\quad \vee \\ &\quad \exists^{\text{out}} y \{y' \leq_{\text{in}} x'\}(\mathbf{o}(y), x) \end{aligned}$$

which could be again simplified into:

$$\forall^{\text{in}} x \{\sigma(x')\}(x) \rightarrow \exists^{\text{out}} y \{y' \leq_{\text{in}} x'\}(\mathbf{o}(y), x)$$

Then, according to all the replacement rules, one gets the \mathcal{L}_D -formula

$$\forall x \{\sigma(x')\}(x) \rightarrow \exists y \{y' \preceq x'\}(y, x)$$

which expresses that for all positions x , if the data type of x is σ , then there is a position y whose data is smaller than that of x .

The converse is slightly easier, since we do not have to deal with inconsistent use of variables. Any \mathcal{L}_D -sentence ψ is converted into an \mathcal{L}_T -sentence ψ' by doing the following replacements:

- any quantifier \exists is replaced by \exists^{out} (any variable is assumed to be quantified over outputs)
- $x \leq y$ is replaced by $x \leq_{\text{out}} y$
- predicates $\{\phi\}(x, y)$ are replaced by $\{\phi'\}(\mathbf{o}(x), \mathbf{o}(y))$ where ϕ' is obtained from ϕ by replacing \preceq by \leq_{in} .

By construction, a typed data word w satisfies ψ iff $\text{t2d}^{-1}(w)$ satisfies ψ' . □

APPENDIX G

SECTION IV: SCOTT NORMAL FORM

Lemma 20. *For any \mathcal{L}_D -formula φ over an alphabet Γ and a type alphabet Σ , one can construct an \mathcal{L}_D -formula ϕ over $\Gamma \times \Gamma'$ and the type alphabet Σ such that:*

- up to projection on Γ , ϕ and φ have the same models,
- ϕ is of the form $\forall x \forall y \psi(x, y) \wedge \bigwedge_{i=1}^m \forall x \exists y \psi_i(x, y)$ where the formulas ψ and ψ_i , $i = 1, \dots, m$, are quantifier free.

Proof. The proof is similar to [17]. We first assume without loss of generality that φ is in negation normal form. We now construct the formula ϕ iteratively. At each iteration, we get formulas θ_i and ϕ_i where φ is equivalent to $\theta_i \wedge \phi_i$,

θ_i is in correct form, and ϕ_i has a number of quantifiers reduced by i compared to φ , while using some additional unary predicates P_1, \dots, P_i . At first let $\theta_0 = \top$ and $\phi_0 = \varphi$. Then, at each step, consider a subformula $\xi_i(x)$ of ϕ_{i-1} with a single quantifier. Then $\xi_i(x)$ is either $\exists y \rho_i(x, y)$ or $\forall y \rho_i(x, y)$ where ρ_i a quantifier free formula. In the first case, we set $\theta_i = \theta_{i-1} \wedge \forall x \exists y (P_i(x) \rightarrow \rho_i(x, y))$ and ϕ_i is obtained by replacing $\exists y \rho_i(x, y)$ by $P_i(x)$.

In the second case, we set $\theta_i = \theta_{i-1} \wedge \forall x \forall y (P_i(x) \rightarrow \rho_i(x, y))$ and ϕ_i is obtained by replacing $\forall y \rho_i(x, y)$ by $P_i(x)$.

This process ends as at each step the number of quantifiers of ϕ_i decreases. In the end, we get ϕ_k which is quantifier free and thus equivalent to $\forall x \forall y \phi_k$. By combining all the double \forall conjuncts into one formula ψ , we finally set $\phi = \theta_k \wedge \forall x \forall y \psi$ which is in the required form. The size of ϕ is linear in the size of the negative normal form of φ . Finally, the unary predicates P_i are added to the alphabet to be treated as letters. This is done by replacing the alphabet Γ by $\Gamma \times \Gamma'$, where $\Gamma' = 2^{P_1, \dots, P_k}$, and replacing in the formula the predicates $P_i(x)$ by the conjunction of letter predicates $\bigvee_{(\gamma, R) \mid P_i \in R} (\gamma, R)(x)$.

We need now to prove the first statement regarding domains. We prove this by induction on the formulas $\theta_i \wedge \phi_i$. Assume that the data word u_i is a model for $\theta_i \wedge \phi_i$, we construct u_{i+1} a model for $\theta_{i+1} \wedge \phi_{i+1}$ by adding truth values for the predicate P_{i+1} by setting $P_{i+1} = \{a_j \mid a_j \text{ is a position of } u \text{ and } (u, j) \models \xi_i(x)\}$. Conversely, if (u_i, P_{i+1}) is a model for $\theta_{i+1} \wedge \phi_{i+1}$, then for any position j of u_i such that $(u_{i+1}, j) \models P_{i+1}(x)$, we also have $(u_i, j) \models \xi_{i+1}(x)$ since P_{i+1} does not appear in ξ_{i+1} . And since φ is in negative normal form, ξ_{i+1} only appears positively and thus $u_i \models \varphi_i$. We conclude by noting that if $(u_i, P_{i+1}) \models \theta_{i+1}$ then $u_i \models \theta_{i+1}$. Notice that the number of predicates added is equal to the number of quantifications in φ and hence is linear. However, since they are not mutually exclusive, this leads to an exponential blow-up of the alphabet Γ' . \square

APPENDIX H FROM \mathcal{L}_D TO MSOLPP

Proposition 22. *From any \mathcal{L}_D -sentence ϕ in SNF over Σ and Γ , one can construct an instance C of MSOLPP such that for any typed data word $w \in \mathcal{TDW}(\Sigma, \Gamma)$, $w \models \phi$ iff $d2p(w) \in \mathcal{LPS}(\Sigma, \Gamma)$ is a solution of C .*

Proof. First, let us remark that due Proposition 21, we can translate a sentence φ from \mathcal{L}_D over $\mathcal{TDW}(\Sigma, \Gamma)$ to an equivalent formula over $\mathcal{LPS}(\Sigma, \Gamma)$ by replacing the order relation by \rightarrow and subformulas $\{\phi\}$ by $\phi[\leq / \leq]$ where \leq is the vertical order. Let $\phi = \forall x \forall y \varphi(x, y) \wedge \bigwedge_{i=1}^n \forall x \exists y \varphi_i(x, y)$ with vertical predicates binary $(\alpha_i)_{i=1}^k$. As mentioned before, we treat 0-ary and unary predicates as binary predicates. Now given x and y , an *atomic type* for x and y gives truth value for the predicates $(\alpha_i)_{i=1}^k$. Formally, it is composed of point labels for x and y , an horizontal direction $x \sim y$ for $\sim \in \{=, \leftarrow, \rightarrow\}$ and truth values for the binary formulas α_i . Then a couple of points (p, q) is of type t if they satisfy exactly

the true properties of t when x and y are evaluated as p and q respectively. Note that any atomic type can be described by a universal constraint using boolean combination of the predicates α_i . Note also that any model of ϕ has to satisfy the universal part $\forall x \forall y \varphi(x, y)$. Hence we want to weed out all atomic types that do not satisfy it. Then the set of universal constraints C_\forall is set as all forbidden types, i.e. the atomic types that do not satisfy $\varphi(x, y)$. Then if w is a typed data word that satisfy ϕ , any pair of positions of w satisfy $\varphi(x, y)$ and any pair of points of $d2p(w)$ will satisfy every constraint of C_\forall . Conversely, if every pair of points of some (L, P) satisfy the constraints from C_\forall , then $d2p^{-1}(L, P)$ will model $\forall x \forall y \varphi(x, y)$.

We now turn to the formulas $\forall x \exists y \varphi_i(x, y)$. By doing an extensive case of study over all atomic types for x and y , and then factorising for each label γ of Γ , we can rewrite the formulas as

$$\forall x \bigwedge_{j=1}^k (\gamma_j(x) \rightarrow \exists y \bigvee_{\ell=1}^m t_{j,\ell})$$

where $t_{j,\ell}$ are atomic types. We conclude depending on the nature of the direction $d_{j,\ell}$ of $t_{j,\ell}$. If $d_{j,\ell}$ is $x = y$, then if $t_{j,\ell}$ is compatible with γ_j the conjunct $\gamma_j(x) \rightarrow \exists y t_{j,\ell}$ is either a tautology and the whole conjunct is trivially satisfied, or it cannot be satisfied and $t_{j,\ell}$ is removed from the disjunction. The remaining elements of the disjunction can be combined in a set E to form an existential constraint with γ_j . Now if $w \models \forall x \bigwedge_{j=1}^k (\gamma_j(x) \rightarrow \exists y \bigvee_{\ell=1}^m t_{j,\ell})$, then for every point p of $d2p(w)$, if p is labelled by γ then there exists a point q such that (p, q) is of one of the type $t_{j,\ell}$ and thus q is a valid witness for p . Conversely, the fact that any point p has a valid witness in (L, P) means that for any position labelled by γ in $d2p^{-1}(L, P)$, there is an other position corresponding its witness that is a valid quantification for y , and thus $d2p^{-1}(L, P)$ will satisfy $\forall x \bigwedge_{j=1}^k (\gamma_j(x) \rightarrow \exists y \bigvee_{\ell=1}^m t_{j,\ell})$.

This gives an instance $C = (C_\exists, C_\forall)$ of constraints over point and line alphabets Σ and Γ such that (L, P) satisfy C if, and only if, $d2p^{-1}(L, P) \models \phi$. \square

APPENDIX I PROFILES

A. Profile of a line

Let us now construct the profile of a line in an LPS and with respect to some MSOLPP instance. Given an LPS $M = (L, P)$ and an MSOLPP instance $C = (C_\exists, C_\forall)$, and an integer $k \leq |L|$, we define the profile $(\sigma, S, A_1 \dots A_n)$ of the k th line of M . S is the set of reachable and co-reachable states of the predicate automata of C , just before position k over the word L and $\sigma = L(k)$. Let i be such that $(i, k') \in \text{dom}(P)$ for some k' , and then let $\gamma = P(i, k')$. If $k' = k$, then let A_i be (γ, \cdot) . If $k < k'$ (resp. $k > k'$), then we define A_i to be the set (γ, R, \uparrow) (resp. (γ, R, \downarrow)) where R is the set of all pairs (p, q) from S_Ψ such that $p \in S$ and there exists an accepting run that reaches p just before position k and q just before position

k' . The symbol \uparrow indicates that the state p eventually reaches q above, and \downarrow that the state q was visited below p . Then we obtain a sequence of clauses $A_1 \dots A_m$ (since the domain of P over the first component is downward closed) but since we want the information of the profile to be bounded, we only keep the leftmost and rightmost occurrence of each clause.

Formally the profile λ_k of the k th line of M is defined as $(\sigma, S, A_{i_1} \dots A_{i_n})$ where $(i_j)_{j \geq 1}$ is an increasing sequence such that $i = i_j$ for some j if, and only if, $A_\ell \neq A_i$ either for all $\ell < i$ or for all $i > \ell$.

B. Consistency

We give here the formal definition of consistency between profiles. Let $\lambda = (\sigma, S, A_1 \dots A_n)$ and $\lambda' = (\sigma', S', A'_1 \dots A'_{n'})$ be two profiles. Let A be a clause of λ . If $A = (\gamma, \cdot)$, then the *successor* of A with respect to λ, λ' is the clause $A' = (\gamma, R, \downarrow)$ with R being the set of pairs (p', p) with $p' \in S', p \in S$ and $p' \in p \cdot \sigma$. If $A = (\gamma, R, \downarrow)$ then its *successor*, for λ, λ' , is the clause $A' = (\gamma, R', \downarrow)$ with R' being the set of pairs (p', q) such that there is a state p such that $(p, q) \in R$ and $p' \in p \cdot \sigma$. If $A = (\gamma, R, \uparrow)$, then the *successors* of A for λ, λ' are either the clause $A' = (\gamma, \cdot)$ if $R = \{(p, q) \in S \times S' \mid q \in p \cdot \sigma\}$, or $A' = (\gamma, R', \uparrow)$ with R' the set of pairs (p', q) such that there is a state p such that $(p, q) \in R$ and $p' \in p \cdot \sigma$. The *predecessors* of a clause with respect to a pair of profiles are defined symmetrically.

We want to define a local consistency for consecutive profiles such that two profiles are consistent if and only if, they can appear in succession in a sequence of profiles of a model. Formally two profiles $\lambda = (\sigma, S, A_1 \dots A_n)$ and $\lambda' = (\sigma', S', A'_1 \dots A'_{n'})$ are consistent if 1) for any state $s \in S$, there is a state $s' \in S'$ such that $s' \in s \cdot \sigma$ and conversely for any state $s' \in S'$ there is a state $s \in S$ such that $s' \in s \cdot \sigma$ and 2) there exist increasing mappings π and π' over integers such that:

- For every $i \leq n$, either there exists j such that $\pi(i) = \pi'(j)$ and A'_j is a successor of A_i (w.r.t λ, λ'), or there exist two indexes k, k' such that $A'_k = A'_{k'}$, $\pi'(k) < \pi(i) < \pi'(k')$ and A'_k is a successor of A_i .
- For every $j \leq n'$, either there exists i such that $\pi(i) = \pi'(j)$ and A_i is a predecessor of A'_j , or there exist two indexes k, k' such that $A_k = A_{k'}$, $\pi(k) < \pi'(j) < \pi(k')$ and A_k is a predecessor of A'_j .

A profile $\lambda = (\sigma, S, A_1 \dots A_n)$ is *initial* if all states in S are initial states. It is *final* if for all states s of S , we can reach a final state by reading σ . A sequence of profiles $\lambda_1 \dots \lambda_n$ is *consistent* if λ_1 is initial, λ_n is final, and for all $i < n$, λ_i is consistent with λ_{i+1} . It is *maximal* if it is consistent and there does not exist a different consistent sequence $\lambda'_1 \dots \lambda'_n$ with the same number of lines and the same number of clauses per line such that for all $k \leq n$, $\sigma_k = \sigma'_k$, $S_k \subseteq S'_k$ and for all i , $A_i^k \subseteq A_i'^k$. Intuitively a consistent profile sequence is maximal if one cannot add states in the lines of the clauses without making it inconsistent.

C. Satisfiability of MSOLPP

Lemma 26. *Given an instance C of MSOLPP, for any model M of C , $\text{Seq}(M)$ is C -valid, consistent and maximal.*

Proof. Let C be an instance of MSOLPP, let M be a model of C and consider $\text{Seq}(M) = \lambda_1 \dots \lambda_n$ the sequence of profiles obtained from M .

Validity. Let $k < n$ and $\lambda_k = (\sigma_k, S_k, A_1^k \dots A_{m_k}^k)$. Since M is a model of C , every existential constraint is satisfied by any point on line k of M . This means that for each such constraint (γ, E) and j such that $A_j^k = (\gamma, \cdot)$, there is an integer i such that (γ, i, k) is a point of M and there exists a triplet $(\gamma', d, \psi_\ell) \in E$, a point (γ', i', k') with $i \sim_d i'$, a pair (p, q) from SP_ℓ and accepting run that reaches p on line k and state q on line k' . By definition of profiles, if such a run exists, then either there is a clause $A_{j'}^k$ of λ_k that contains $(\gamma', (p, q), v)$, or there exist smaller and greater positions with the same set of runs, and thus containing it. In both cases, we have a clause $A_{j''}^k$ in λ_k with $j \sim_d j''$ which is consequently a valid witness for the point. Thus λ_k satisfies all existential constraints. Now if λ_k fails a universal constraint $(\gamma, \gamma', d, \psi_\ell)$, it means that there are j and j' such that $j \sim_d j'$, $A_j^k = (\gamma, \cdot)$ and $A_{j'}^k$ contains (γ', p, q, v) for some $p, q \in SP_\ell$ and v a vertical direction. This means that there exists a point (γ', i'', k') in M from which originates $A_{j'}$. By construction of the predicate automata, it means that the corresponding pair of points fails the universal constraint $(\gamma, \gamma', d, \psi_\ell)$, which contradicts our assumption. Thus λ_k satisfies the universal constraints, and consequently each λ_k is valid.

Consistency. By construction, all clauses of the profiles λ_k describe accepting runs. In particular, all states of λ_1 are initial, and all states of λ_n lead to final states. Now consider $k \leq n$ and A_i^k an element of λ_k . Then A_i^k describes partial accepting runs over $\sigma_1 \dots \sigma_n$, and as these runs do not depend on k , they also appear on line $k+1$. Then either they can be found in λ_{k+1} , or they have been deleted, which by construction imply that the same set of runs already appear in clauses both before and after i . This means that A_i^k has a successor in λ_{k+1} . Using symmetric arguments, every clause in λ_{k+1} has a predecessor in λ_k , thus the sequence is consistent.

Maximality. If $\lambda_1 \dots \lambda_n$ is not maximal, then there exist clauses that are not maximal. It implies that there is at least one accepting run that does not appear in its own clause, even though it does not appear between two similar runs described by other clauses. This is a contradiction since the very definition of profile states that the only reason a piece of information can be forgotten is that it appears in clauses before and after the position. \square

Lemma 27. *Given an instance C of MSOLPP and a valid, consistent and maximal sequence of C -profiles $s = \lambda_1 \dots \lambda_n$, there exists a model M of C such that $s = \text{Seq}(M)$.*

Proof. Let C be an instance of MSOLPP and $\lambda_1 \dots \lambda_n$ be a valid, consistent and maximal sequence of profiles. Our first goal is to construct a system $M = (L, P)$ such that the

$a', \{q_y\}$	$(b, \binom{(q_y, q_x)}{(q_y, q_{a'})}, \downarrow)$	$(a, (q_y, q_x), \downarrow)$	(b, \cdot)	$(b, (q_y, q_{a'}), \downarrow)$	(c, \cdot)	$(a, (q_y, q_x), \downarrow)$
$a', \{q_{a'}\}$	$(b, \binom{(q_{a'}, q_x)}{(q_{a'}, q_{a'})}, \downarrow)$	$(a, (q_{a'}, q_x), \downarrow)$	$(b, (q_{a'}, q_y), \uparrow)$	(b, \cdot)	$(c, (q_{a'}, q_y), \uparrow)$	$(a, (q_{a'}, q_x), \downarrow)$
$c', \{q_x, q_{a'}\}$	(b, \cdot)	$(a, \binom{(q_x, q_x)}{(q_{a'}, q_x)}, \downarrow)$	$(b, \binom{(q_x, q_y)}{(q_{a'}, q_y)}, \uparrow)$	$(b, \binom{(q_x, q_{a'})}{(q_{a'}, q_{a'})}, \uparrow)$	$(c, \binom{(q_x, q_y)}{(q_{a'}, q_y)}, \uparrow)$	$(a, \binom{(q_x, q_x)}{(q_{a'}, q_x)}, \downarrow)$
$b', \{q_x\}$	$(b, \binom{(q_x, q_x)}{(q_x, q_{a'})}, \uparrow)$	(a, \cdot)	$(b, (q_x, q_y), \uparrow)$	$(b, (q_x, q_{a'}), \uparrow)$	$(c, (q_x, q_y), \uparrow)$	(a, \cdot)

Fig. 4. Profile sequence of the LPS from Figure 2, with $Bet_{a'}$ predicate from Example 23.

sequence of profiles $\text{Seq}(M)$ associated with M is $\lambda_1 \dots \lambda_n$. The main difficulty is to spatially place all clauses (γ, \cdot) in a coherent fashion. Then, we show that M is a model of C thanks to the properties of the sequence $\lambda_1 \dots \lambda_n$.

We aim to construct $M = (L, P)$. For $k \leq n$, let $\lambda_k = (\sigma_k, S_k, A_1^k \dots A_{m_k}^k)$. The first part, L , is already completely defined by the sequence $\lambda_1 \dots \lambda_n$ by definition of profiles. We set $L(k) = \sigma_k$. Regarding the set of points P , for each clause of the form $A_i^k = (\gamma, \cdot)$, we create a γ -labelled point (γ, j, k) in the two-dimensional space, for some well-chosen abscissa j (we say that we *place* the clause A_i^k). These choices should ensure that we get exactly the profile sequence $\lambda_1 \dots \lambda_n$. To this end, we virtually place all clauses in the two-dimensional space, with the idea that two clauses in λ_k and λ_{k+1} that follow each other will appear at the same abscissa. Placing all the clauses A_i^k is done by defining a mapping $\eta : (i, k) \mapsto j$, which will be used after to show that all the constraints of C are satisfied. The set of points P is then defined as $P = \{(\gamma, j, k) \mid A_i^k = (\gamma, \cdot) \text{ and } \eta(k, i) = j\}$. Intuitively, the mapping η stems from the mappings π_k and π'_k describing the consistency of profiles λ_k and λ_{k+1} . These two mappings induce a total pre-order on the clauses of λ_k and λ_{k+1} , which is exactly what η aims to do, but generalised to all the clauses.

Taking the transitive closure of the union of all such pre-orders defines a pre-order \leq over all clauses. The mapping η is then defined as one possible linearisation of the pre-order.

The relation \leq is defined on $\{k, m \mid 1 \leq k \leq n, 1 \leq m \leq m_k\}$ as the transitive closure of the pre-orders induced by the mappings π_k and π'_k for all $k < n$. Formally, we have $(k, m) \leq (k', m')$ if

- $k = k'$ and $m \leq m'$,
- $k < k'$ and there exists $(m_\ell)_{\ell=k}^{k'}$ such that $m_k = m$, $m_{k'} = m'$ and for all $k \leq \ell < k'$, $\pi_\ell(m_\ell) \leq \pi'_\ell(m_{\ell+1})$,
- $k > k'$ and there exists $(m_\ell)_{\ell=k'}^k$ such that $m_k = m$, $m_{k'} = m'$ and for all $k' \leq \ell < k$, $\pi_\ell(m_\ell) \leq \pi'_\ell(m_{\ell+1})$.

The relation \leq is clearly a pre-order. Then we can arbitrarily define a mapping $\eta : (\{k, m \mid 1 \leq k \leq n, 1 \leq m \leq m_k\}, \leq) \rightarrow (\mathbb{N}, \leq)$ that preserves the order.

Let us now show that the sequence of profiles of (L, P) $\lambda'_1 \dots \lambda'_{n'}$ is equal to $\lambda_1 \dots \lambda_n$. First, it is easy to see that the two sequences have the same number of lines n . They also have the same line labels σ_k for $1 \leq k \leq n$. Consequently, for each line k the sets of available states S_k is the same for both sequences. Now consider a clause A_i^k from profile λ_k . It describes runs to a clause $A_{i'}^{k'}$ of the form (γ, \cdot) that is

either minimal or maximal. Since $(\gamma, \eta(k', i'), k')$ is a point of P , this clause also appear in (L, P) and since the points of P stem from clauses of the sequence $\lambda_1 \dots \lambda_n$, this clause is also either maximal or minimal in λ_k , and thus the clause appears in $\lambda'_1 \dots \lambda'_n$. Conversely, clauses of λ'_k are induced by maximal or minimal runs to a point of P , all of which originate from clauses of the form (γ, \cdot) from $\lambda_1 \dots \lambda_n$, ensuring that they appear in λ_k . This proves that λ_k and λ'_k have the same clauses. The fact that η preserves the pre-order \leq and the order $<$ prove that $\lambda_k = \lambda'_k$, completing the equality.

We only have left to show that (L, P) is indeed a model of C . Let (γ, E) be an existential constraint of C and (γ, i, k) be a point of P . By construction, this means that there exists i such that $A_i^k = (\gamma, \cdot)$. Since λ_k is valid, there exists a tuple (γ', d, ψ) of E and $j \leq n$ such that $i \sim_d j$, and (p, q) in SP_ψ and $v \in \{\uparrow, \downarrow\}$ such that $(\gamma', p, q, v) \in A_j$. Moreover, since $\lambda_1 \dots \lambda_n$ is a consistent sequence, there exist k' and j' such that $A_{j'}^{k'} = (\gamma', \cdot)$ and $\eta(k, j) = \eta(k', j')$. The clause A_j then describes an accepting run to the point $(\gamma', \eta(k', j'), k')$ in P and consequently this point is a valid witness of (γ, i, k) . Thus (L, P) is valid with respect to the existential constraints of C .

Now consider a universal constraint $(\gamma, \gamma', d, \psi)$. If (L, P) does not satisfy it, then there exist two points (γ, i, k) and (γ, i', k') such that $i \sim_d i'$ and there exist p, q in SP_ψ and an accepting run on L that reaches state p just before position k and state q just before position k' . By construction of (L, P) there exists j (resp. j') such that $A_j^k = (\gamma, \cdot)$ (resp. $A_{j'}^{k'} = (\gamma', \cdot)$). Since the input sequence is maximal, and since (γ', p, q, v) is a valid clause for λ_k either there exists ℓ such that $\eta(k, \ell) = \eta(k', j')$ or there are two indexes, one greater and one smaller, that have this clause. In both cases, we can find a clause in λ_k such that λ_k does not satisfy $(\gamma, \gamma', d, \psi)$, which contradict the initial assumption. Then (L, P) satisfies $(\gamma, \gamma', d, \psi)$ and thus is valid. \square

Lemma 28. *Given an instance C of MSOLPP, the set $\{\text{Seq}(M) \mid M \models C\}$ is effectively regular.*

Proof. By Lemmas 26 and 27, the set $\{\text{Seq}(M) \mid M \models C\}$ is exactly the set of profile sequences that are valid, consistent and maximal. We construct an automaton that accepts valid, consistent and maximal sequences of C -profiles. Let us denote by Π_C the set of valid profiles. Then we define the automaton $A = (Q, \mathcal{P}, I, \Delta, F)$ where $Q = \Pi_C$, I (resp. F) is the set of valid initial (resp. final) profiles, and $\Delta \subseteq \mathcal{P}_C \times \Pi_C \times \Pi_C$ as $\Delta = \{\lambda \xrightarrow{\lambda'} \lambda' \mid \lambda' \text{ is valid and } \lambda \text{ is consistent with } \lambda'\}$.

We modify this deterministic automaton to accept only maximal sequences by allowing the automaton to non-deterministically guess a run that could be added to the sequence. This gives an automaton that accepts non maximal sequences, and we conclude since recognisable languages are closed by symmetric difference. \square

APPENDIX J OTHER RESULTS

Theorem 29. *Let C be an instance of MSOLPP and $\llbracket C \rrbracket$ be its set of valid models. Then:*

- *The set of vertical words of $\llbracket C \rrbracket$ is regular.*
- *It is decidable if all models of $\llbracket C \rrbracket$ have bounded line width.*
- *It is decidable if there exist two different models with the same vertical word.*
- *There is $k \in \mathbb{N}$ such that to any vertical word of $\llbracket C \rrbracket$ one can associate a model of line width at most k .*

Proof. We prove the four statements separately.

- The regularity of the vertical language of $\llbracket C \rrbracket$ is a direct consequence of Lemma 28, since the profiles hold the information regarding line labels and regularity is preserved by projection.
- This is also a consequence of the equivalence with profiles. Indeed, the very definition of profile states that if a given letter appear twice or more on a given line, the corresponding profile is the same. Conversely, if each letter appear at most once on every line, then the lines of the models have bounded width. Thus this question amounts to decide if there exists a valid sequence of profile where at least one profile has a letter which is repeated. This can be decided by checking if there exists an accepting run in the automaton from Lemma 28 where such a profile appear.
- Assume that $\llbracket C \rrbracket$ is of bounded line width, otherwise it is trivially false due to the pumping argument of the previous item. We prove that if there exists two models with the same vertical word, then we can find some within some bounded height. Indeed, it either amounts to find two runs of the profile automaton \mathcal{A} for valid sequences that have different sequences of profiles or find two different models having the same valid sequence of profiles. The former case amounts to check if there are two different sequences of profiles with the same vertical word that are accepted by \mathcal{A} . This can be done by determinizing it, projecting it on the vertical labels, and check for unambiguity. For the latter case, consider two different models (L, P) and (L, P') and let $P_1 \dots P_n$ be their common sequence of profiles. If n is big enough, then there exists an accepting run r of \mathcal{A} over $P_1 \dots P_n$ and integers $i_1 < i_2 < i_3 < i_4 \leq n$ such that r reaches positions i_j , for $1 \leq j \leq 4$, in the same state. Let now ℓ be the smallest abscissa such that the points (γ, ℓ, k) and (γ', ℓ, k') are different. Then there exists $j \leq 3$ such that $k, k' \notin [i_j, i_{j+1}]$. Let us prove that we can shorten (L, P) and (L, P') to get two different models with the same

profile and whose length is smaller than n . The idea is to suppress from P and P' all points whose line is between i_j and i_{j+1} . Then by resetting the ordinate of points above i_{j+1} and using a shrinking function $\eta : \mathbb{N} \rightarrow \mathbb{N}$ for abscissa, we get downward closed domains and hence smaller models (K, R) and (K, R') whose sequence of profiles is in both cases $\lambda_1 \dots \lambda_{i_j-1} \lambda_{i_{j+1}} \dots \lambda_n$ which is accepted by \mathcal{A} by construction. But since ℓ was the smallest abscissa on which the two labelled sets points disagree, the operation of suppressing the lines between i_j and i_{j+1} have the same effect on all points between i_j and i_{j+1} with abscissa smaller than ℓ , i.e. such points are either preserved or suppressed in both models simultaneously. This implies that the models R and R' still disagree on the abscissa $\eta(\ell)$. Hence we can check functionality by checking it on bounded-width sequence of profiles of length smaller than $4 \cdot |Q_{\mathcal{A}}|$.

- Since profiles have bounded width then, according to Lemma 27, one can associate to any word of the vertical domain a model of bounded width. Given a word of the vertical domain u of the profile automaton, one can associate a sequence of profile s whose projection on Σ^+ is u . Then by Lemma 27, we can concretize s into a model M that has n points, where n is the number of (γ, \cdot) clauses appearing in s . Now since each clause appear at most twice in a given profile, the width of a line of M is bounded by $2 \cdot |\Gamma|$. \square

Corollary 30. *Given an \mathcal{L}_T -transduction τ :*

- *The domain of τ is regular.*
- *It is decidable whether τ has bounded origin.*
- *It is decidable whether τ is functional.*
- *τ has a bounded origin uniformisation.*

Proof. Clearly, the construction from Proposition 17, transforming a transduction into a non-erasing one by adding a copy of the input, preserves these three properties, although the bound on origin is incremented by 1. Also, since the correspondence with typed data word is one-to-one (Theorems 19), they are also transferred to \mathcal{L}_D . This is also the case when transferring to LPS, together with the logic with directions, and from formulas with quantifier depth at most 2 to sets of constraints. The critical part is the SNF. Here, the domain is not preserved per se since the models are enriched with unary predicates. However, the truth value of these new predicates is not arbitrary as they stem from the truth value of positive sub-formulas of the input. Then there is a canonical and bijective way to transform valid models for the input formula to valid models for the normalised one, since the opposite transformation is just the projection. Finally, since, for a given instance of MSOLPP, we can transform valid LPS to valid sequences of profiles back and forth, the regularity of the input domain, functionality of a transduction, bounded origin of a transduction and having a bounded origin uniformisation for a transduction translate in the MSOLPP problem as follows:

- The regularity of the input domain is equivalent to the regularity of the vertical domain.
- It is functional if there does not exist two valid models with the same vertical word.
- The bounded origin is equivalent to the bounded size of the lines of valid models.
- Having a bounded origin uniformisation is equivalent to the profiles being of bounded size.

□